Universidad de las Ciencias Informáticas

Facultad 5

TRABAJO DE DIPLOMA EN OPCIÓN AL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Módulo de estados de juego y componentes de presentación para el motor de videojuego Cneuro Game Engine

Autor:

Dania Ramírez Mora

Tutor:

Ing. Yenifer del Valle Guevara

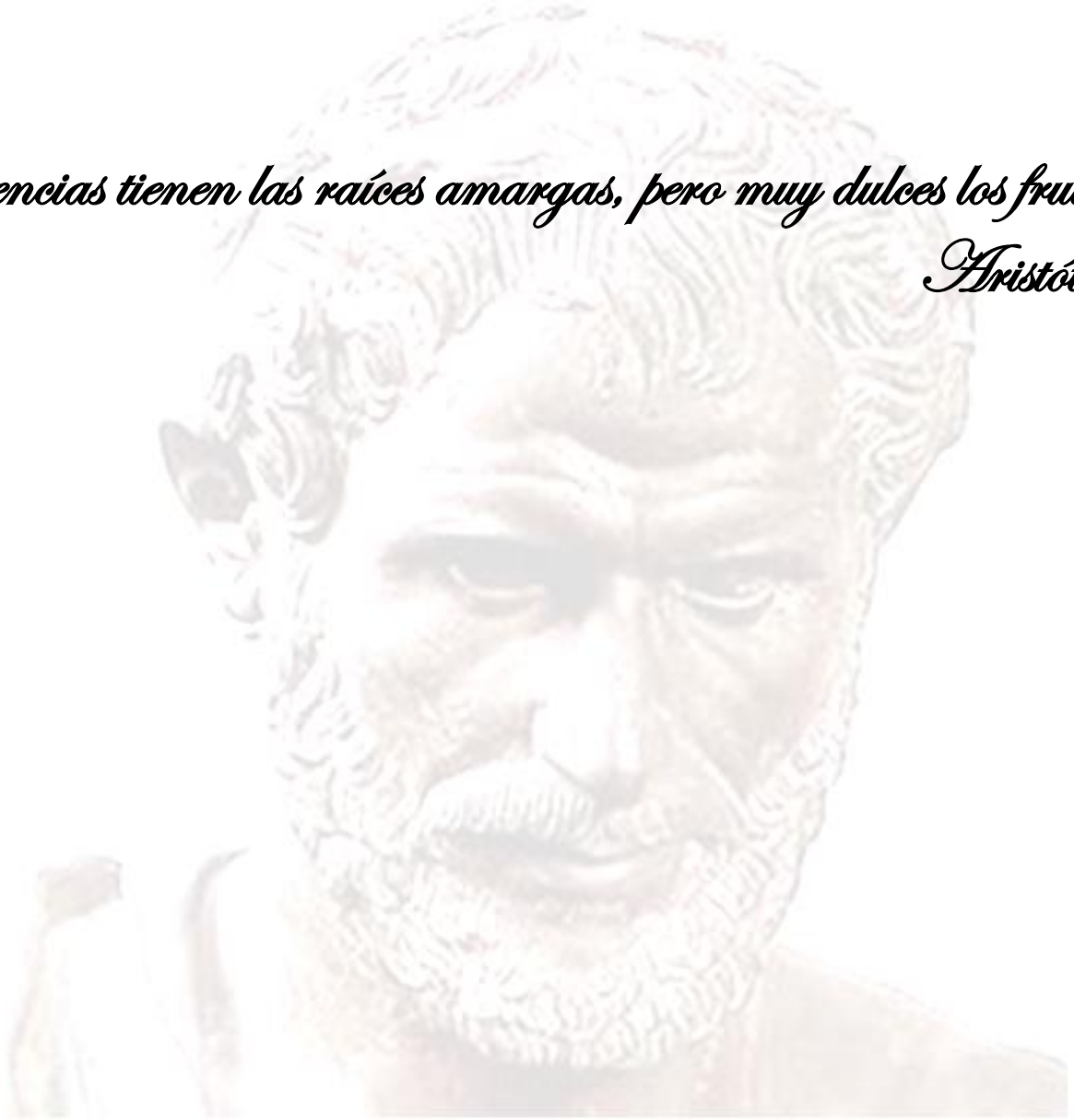
Ciudad de La Habana

2010

¨ Año 52 de la Revolución ¨

"Las ciencias tienen las raíces amargas, pero muy dulces los frutos".

Aristóteles.



Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

"[Insertar nombre(s) de autor(es)]"

_____

"[Insertar nombre(s) de tutor(es)]"

_____

Agradezco a mi madre, el ser más hermoso que haya conocido en la vida, el más grande de mis amores que con dedicación y consagración sin igual ha hecho de mí la persona que soy hoy. Siempre con el consejo preciso, el regaño oportuno, el beso afectuoso, la preocupación constante y el cariño inmenso. A ti mamita querida van ofrendados todos mis triunfos.

A mis hermanos Diana y Damián por estar a mi lado en el momento preciso, por brindarme el calor de hermano que tanto me alentó cuando me sentía caer, por elevar el concepto de familia a su más alta expresión.

A una persona que me acogió como su hija ofreciéndome su amor y cariño: Gonzalo. A Odalis por sus acciones, por sus detalles, por tanta preocupación y ternura.

A mi familia en general por la confianza depositada en mí: mi papá Edilberto, a mis tíos Delio y Mindalia, a mis primas Anabel y Elizabeth.

A mi tutora Yenifer por su ayuda certera y su completa disposición cuando lo necesité.

A todos los profesores implicados en mi formación estudiantil por su entrega y derroche de sabiduría, en especial a Rafaela González y Yadira Ramírez.

A mi tribunal por sus sabios consejos y por guiarme en el desarrollo de este trabajo.

A mi compañero del alma, mi novio Ernesto, por su amor, su fiel compañía, sus enseñanzas, por su perdón ante el error cometido, por darme tanta felicidad y hacer inolvidables mis años de estudiante universitaria.

A mis queridas amigas siempre solícitas ante mis llamadas, por brindarme su ayuda incondicional, su total comprensión, y por ser como hermanas sobre todas las cosas: Ivelisse, Dollys, Yunetsy y Denia.

A alguien que desinteresadamente me ha regalado su cariño y ternura: Cristina.

A mis vecinos por su preocupación constante.

A nuestro comandante en jefe por haber ideado este centro de altos estudios .A la Revolución cubana por permitir que muchos estudiantes al igual que yo se formen profesionalmente sin costo alguno.

A la memoria de mis abuelitos Fausto y María

A mi madre por darme la vida

A mis hermanitos queridos

A mi familia, mi más preciado tesoro

A mi novio por su amor incondicional

A mis amigos

Resumen

En el proceso de realización de un videojuego es de sumo interés, si se desea lograr un funcionamiento loable del mismo, brindar especial atención a la organización de las tareas y eventos en estados independientes que se ejecuten cuando sean invocados sin restarle importancia además al modo en que se presenta la información, que resulta un aspecto significativo para conquistar el éxito del videojuego si de atracción y preferencia se habla, de ahí que se deba diseñar una interfaz de usuario coherente, sencilla y útil unida a un buen sistema de entrada que permita al jugador familiarizarse deprisa con el funcionamiento del videojuego y obtener su máximo rendimiento.

En el presente trabajo se realiza un estudio y resolución de la necesidad que posee el motor de videojuego Cneuro Game Engine de incorporarle un módulo de estados de juego y componentes de presentación que posibilite un fraccionamiento y estructuración lógica de las funcionalidades del motor en estados de juego y permita ofrecer a los usuarios finales una fácil navegabilidad entre las diferentes pantallas del videojuego así como toda la información requerida en la presentación [1] del mismo de forma eficiente y las vías para interactuar de la manera más factible con él.

Palabras Claves: estados de juego, motor de videojuego, presentación.

[1] Dígase de todos los componentes visuales que introducen al jugador en el ambiente del juego. Incluye la teoría de interfaz gráfica de usuario.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

Acorde con el diccionario de la Real Academia Española, videojuego es un “dispositivo electrónico que permite, mediante mandos apropiados simular juegos en las pantallas de un televisor u ordenador”. Esta definición posee algunas incongruencias que la apartan un tanto del concepto técnico del término videojuego, pudiendo entonces conceptualizarse como un programa informático, normalmente asociado a un hardware específico, que recrea un ejercicio sometido a reglas, se debe lograr uno o varios objetivos y donde los jugadores pueden interactuar y tomar decisiones.

El desarrollo y evolución de este famoso software data del año 1943, cuando la idea de un videojuego fuera concebida y patentada por Thomas T. Goldsmith Jr. y Estle Ray Mann, aunque en realidad se materializara dos décadas más tarde por un físico estadounidense llamado William Higinbotham quien diseñara un sistema de juego virtual que simulaba al tenis nombrado *“Tenis for Two”.* En los años sucesivos la industria de los videojuegos fue transitando por un proceso escalonado de desarrollo que derivó en la producción de videojuegos más sofisticados y cercanos a nuestra realidad cotidiana, incrementándose considerablemente el índice de adeptos a esta forma de entretenimiento en la población.

Detrás de toda esa gama de colores, divertidas animaciones y singulares combinaciones de sonidos que conforman un videojuego y que análogamente a una maquinaria serían las piezas que se engranan para formarla, existe toda una compleja arquitectura que le es transparente al usuario. Dicha arquitectura habitualmente es estándar para la mayoría de los videojuegos. Dentro de sus componentes hay dos en particular que aunque no estén concebidos entre los medulares, guardan especial interés: el gestor de estados y la presentación del videojuego.

El gestor de estados encapsula la lógica del juego. Se resume en el fraccionamiento de las funcionalidades del videojuego en diferentes estados de juego que actúan como módulos independientes y que se supeditan a un administrador de estados que es el encargado de indicar cual estado es el que está activo y monitorizar las transiciones entre ellos. Su inclusión en el videojuego es de necesidad obligatoria para que el jugador pueda manipularlo a su comodidad, por ejemplo: pausarlo, continuar jugando, retornar al menú principal o salir del videojuego.

La bibliografía especializada en el ámbito de los motores de videojuegos no brinda una concepción científica acerca del término presentación. En el contexto de la presente investigación representa el conjunto de componentes visuales, que se muestran antes del comienzo de la partida y que le dan introducción al videojuego brindando al jugador el control del mismo y la información necesaria para la realización de las acciones subsiguientes, dígase video introductorio y las ventanas o pantallas de juego con sus correspondientes menús y gramáticas visuales. La teoría de interfaz gráfica de usuario es el elemento sustancial en el concepto de presentación, sin ella no podría concebirse.

Cuba, canalizando su esfuerzo en el desarrollo de la industria del *software*, se ha comenzado a insertar a esta vasta comunidad mundial de desarrolladores de videojuegos. El surgimiento de la Universidad de las Ciencias Informáticas (UCI) en el año 2002, la cual se concibió desde sus inicios como el bastión de la informatización de la sociedad cubana y en la que se ha venido aplicando un modelo sui géneris de formación donde se vincula la docencia con el proceso productivo, ha revolucionado esta rama de la producción de *software*.

El proyecto de Investigación y Desarrollo(I+D) Juegos Cneuro, perteneciente a Línea de Núcleo Gráfico del Centro de Desarrollo de Informática Industrial de la Facultad 5, el cual desarrolla el motor de juego Cneuro Game Engine para la creación de videojuegos en tres dimensiones (3D) con fines terapéuticos, actualmente enfrenta un problema real y es la no posesión de un **módulo de estados de juego y componentes de presentación** que le viabilice al usuario una loable comunicación con el videojuego y que realice la segmentación del motor de en estados de juego, trayendo como consecuencia el entorpecimiento del curso normal de eventos que debe seguir este tipo de *software,* la lógica de su funcionamiento, así como la adecuada fragmentación y organización de las tareas en estados de juego que posibilitan una factible navegación del jugador a través del videojuego. Agregando a lo anterior que la carencia de una eficiente presentación dificulta el fácil aprendizaje y uso del videojuego para el jugador, por lo que dicho módulo resulta indispensable para lograr un producto con la calidad requerida que funcione en consecuencia a los requisitos propuestos.

Partiendo de la situación problemática esbozada con anterioridad se expone como **problema científico** ¿Cómo propiciar una factible navegabilidad e interacción usuario- juego más amigable para el Cneuro Game Engine?

Teniendo en cuenta el problema científico perfilado se formula como **objeto de estudio** el gestor de estados de juegos y presentación en motores de videojuegos siendo su Campo de acción los estados de juego y componentes de presentación en el motor de juego Cneuro Game Engine.

Se concreta como **objetivo general** de este trabajo de diploma desarrollar un módulo de estados de juego y componentes de presentación conforme a los requerimientos del Cneuro Game Engine.

Tras haber planteado el objetivo que guiará este estudio, se trazan las siguientes **tareas de investigación** que posibilitarán el cumplimiento del mismo:

- ✓ Estudio sobre las tendencias del diseño de interfaces de los videojuegos en el mundo y en Cuba para obtener una visión del estado del arte respecto a este tema que permita modelar el proceso de solución al problema existente.
- ✓ Estudio sobre el funcionamiento e implementación de los estados de un videojuego para comprender su mecanismo de funcionamiento.
- ✓ Diseño del módulo de estados de juego y componentes de presentación del Cneuro Game Engine.
- ✓ Implementación de los componentes básicos del módulo de estados de juego y componentes de presentación del CNeuro Game Engine.
- ✓ Integración del módulo de estados de juego y componentes de presentación al motor de videojuego Cneuro Game Engine.

Para materializar las tareas de investigación recién expuestas y realizar un razonamiento analítico del objeto de estudio se hizo uso de los siguientes métodos científicos:

El Analítico-Sintético para extraer los elementos, teorías y conceptos más significativos sobre la presentación y estructuración de los estados de juego, en los videojuegos que se producen actualmente a nivel mundial que no es más que el objeto de estudio que ocupa al presente trabajo.

El histórico–lógico posibilitó comprobar en teoría la evolución y desarrollo de los videojuegos desde su surgimiento en el siglo pasado, enfocando la atención en cómo se han comportado el diseño de interfaces visuales y del gestor de estados de juego.

El presente documento se estructura en resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, bibliografía referenciada, bibliografía consultada, glosario de términos general, glosario de abreviaturas y anexos.

El Capítulo 1 nombrado Fundamentación Teórica constituye una disertación sobre la función que desempeñan dos componentes fundamentales en la arquitectura y la teoría del funcionamiento de los motores de videojuegos: el gestor de estados y la presentación. Se aborda además el estado del arte y la actualidad nacional, específicamente en la Universidad de las Ciencias Informáticas, referente al tema en cuestión.

El Capítulo 2 titulado Características del sistema, engloba lo concerniente a la descripción del dominio del sistema sintetizando sus conceptos fundamentales. Se realiza la captura de requisitos funcionales y no funcionales, a partir de los cuales se identifican los fragmentos de funcionalidades del sistema en términos de casos de uso.

El Capítulo 3 denominado Diseño e Implementación de la solución**,** aborda todo el proceso de construcción del sistema, comenzando por el modelado del diseño, que describe la forma en que se implementará la aplicación a través de los diagramas de clases y de secuencias y culminando con la ejecución de las actividades sustanciales del flujo de trabajo implementación, el cual arroja como resultado el modelo de implementación que constituye la composición física de la implementación del sistema.

Capítulo 1: Fundamentación Teórica.

1.1. Introducción

La disciplina de creación de motores de videojuegos, desde su surgimiento en la década de los años noventa del pasado siglo, ha sufrido una metamorfosis total si de sofisticación y complejidad se habla; centrándose fundamentalmente en facilitar el proceso de estandarización de los videojuegos. El continuo avance del hardware, específicamente de la tarjeta gráfica, ha posibilitado que varias empresas emergieran proponiendo soluciones que con más o menos éxito, han conseguido que hoy día se pueda disfrutar en los videojuegos de una calidad técnica alucinante.

La programación de estos motores se basa en el principio de la modularidad con el objetivo de dividir el programa en varios módulos o partes diferentes donde cada uno resolverá un fragmento del problema y luego dichos módulos interactuarán entre sí. De esta forma se disminuye el grado de dificultad y se logra una fácil comprensión de terceras personas, facilitando así una mayor reutilización de sus componentes.

Dado que el módulo de estados de juego y componentes de presentación que ocupa a este trabajo, se integra al motor de videojuego Cneuro Game Engine, a lo largo del presente capítulo se pretende demostrar la función que desempeñan los dos elementos vitales que forman a este módulo: gestor de estados y presentación, en la dinámica del funcionamiento de un motor de videojuegos. Todo ello mediante la descripción de conceptos fundamentales como lo son: motor de videojuegos, gestor de estados, estados de juego, motor de interfaz e interfaz gráfica de usuario. Usualmente en la mayor parte de los motores de videojuegos estos dos componentes suelen estar separados en módulos diferentes pero para el contexto de este trabajo se han fusionado por resultar más ajustable a las características del **motor de videojuego** CneuroGameEngine.

1.2. Motor de videojuego.

Los videojuegos inicialmente se creaban escribiendo el código completo; pero su complejidad ha crecido tanto que ya no es posible utilizar esta técnica y se han creado núcleos de videojuegos de código modular. Habitualmente los núcleos de videojuegos se diseñaban para un juego específico, pero se hacían tan generales que podían utilizarse para videojuegos de la misma familia. (1)

Un motor de videojuego o *Game Engine* es el núcleo de un videojuego. Hace referencia a una serie de rutinas que permiten la ejecución de todos los elementos del juego. Es donde se controla cómo se representa cada elemento del juego y cómo se interactúa con ellos. Se gestiona la Inteligencia Artificial (IA), comportamiento, personalidad, habilidad de los elementos del juego, los sonidos asociados a cada elemento del juego en cada momento y todos los aspectos gráficos asociados a estos, incluida la cinemática de éste. Se puede decir que el motor de videojuego equivale a una conjunción del Motor Gráfico, Motor de Sonido, Motor de IA y Motor Físico (modela los aspectos físicos de nuestro mundo virtual como explosiones, líquidos, gravedad, partículas, etc.), más las reglas necesarias para crear el universo completo del juego. Un motor de videojuego puede plantearse para dar soporte a una solución en concreto o para acaparar un dominio mayor. (2)

Las funcionalidades que puede brindar es una de las decisiones más importantes que habrá que tomar. Serán diferentes según el género del videojuego y deberá decidirse hasta que nivel serán ofrecidas. Entre ellas cabrá especificar diferentes aspectos como por ejemplo:

- Desde el punto de vista gráfico qué tipo de escenario se puede tratar interiores, exteriores o genéricos; y con qué tipo de recurso o formato se trabajará para texturas o modelos 3D.
- Aspectos de inteligencia artificial brindados: scripting, comportamientos (behaviours), búsqueda de caminos, etc.
- Qué tema de física se abarcará: estructuras de descomposición espacial como BSP u octrees, reacción a impactos, fuerzas elásticas e inelásticas, simulaciones, etc.

1.3. Arquitectura de un Motor de Videojuego

La arquitectura de un motor de videojuego define el esqueleto de la aplicación. El motor es dividido en módulos para esquematizar con más claridad su funcionamiento.

En la arquitectura de un motor de videojuego se pueden distinguir los siguientes módulos principales:

- Motor gráfico.
- Motor lógico (scripting, comportamientos).
- **Motor de interfaz.**
- **Gestor de estados**
- Motor de datos (niveles, gráficos, etc.).
- Tratamiento de eventos.
- Motor físico.
- Capa de red.
- Motor de sonido (Sonido 3D y efectos especiales).

Además, se pueden encontrar otros módulos secundarios como pueden ser:

- Sistema de configuración.
- Fuentes y textos.
- Sistema de menús.
- Sistema de ayuda.
- Música. (2)

A continuación se ilustra la relación existente entre los módulos mencionados con antelación.

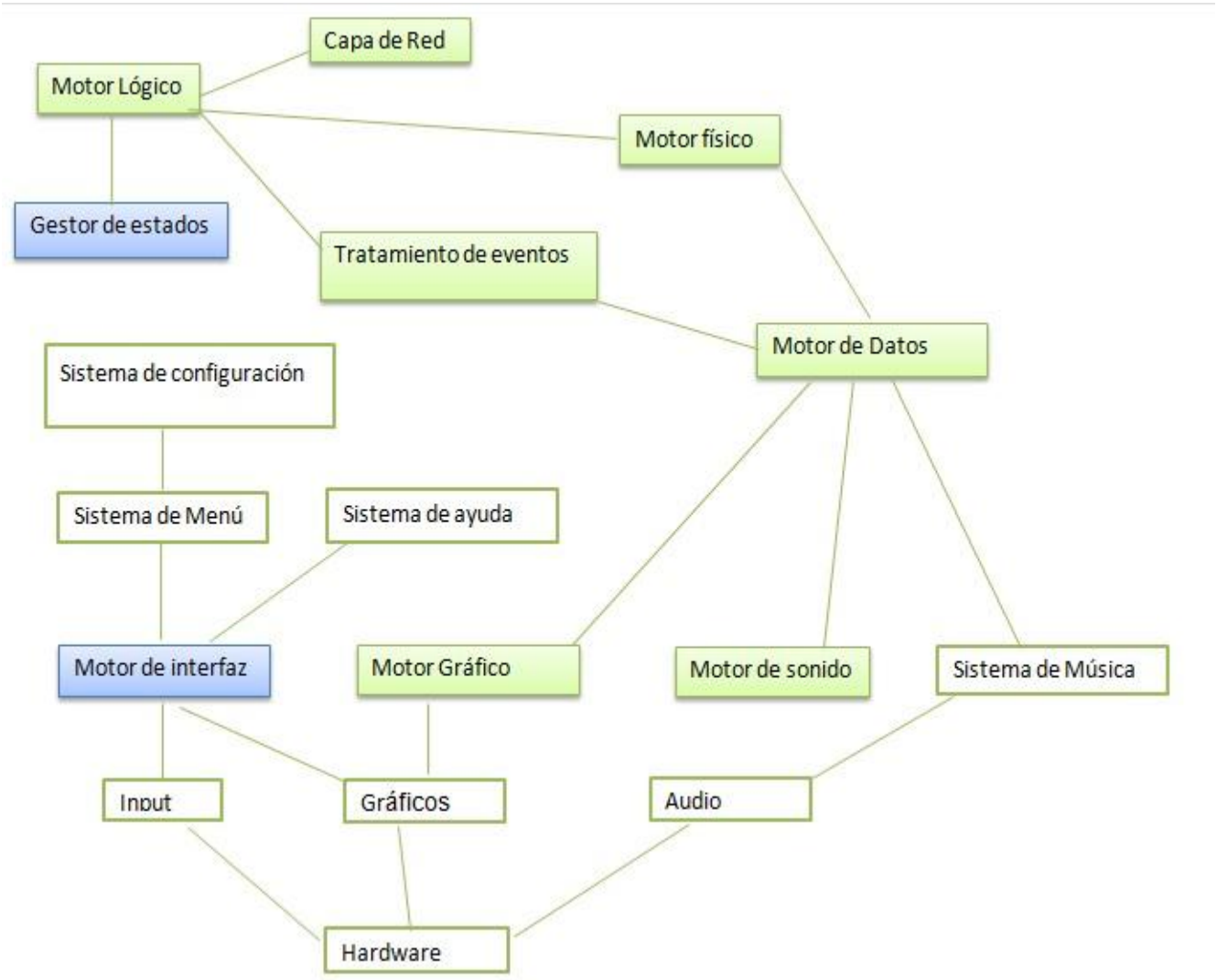


Fig. 1 Arquitectura de un motor de videojuego

Particularmente son de interés para esta investigación dos módulos en específico de la arquitectura que se expone: el gestor de estados y el motor de interfaz, este último debido a la conexión que posee con la presentación, ya explicada en apartados anteriores. Cabe entonces formular la siguiente pregunta: ¿qué rol desempeñan en la dinámica del funcionamiento del motor de videojuego el gestor de estados y el motor de interfaz? Los sucesivos epígrafes darán respuesta a la interrogante planteada.

1.4. Gestor de estados

Es un sistema que posibilita el encapsulamiento de las funcionalidades y eventos del videojuego en estados, que se comportan como entidades autónomas pero que a la vez están bajo el control del administrador de estados que les indica en qué momento y a quien, le corresponde entrar en acción. Agrupa cuatro elementos esenciales: estados de juego, transiciones entre estados, reglas o condiciones y eventos de entrada. Generalmente se implementa mediante una máquina de estados finita o diagrama de estados.

1.4.1. Estados de juego

Un juego no se desarrolla completamente desde el principio (Inicialización) hasta el final (Acciones post-juego) de su secuencia lógica. Durante el juego, es posible encontrarnos en diferentes estados.Se entiende por estado de juego un módulo independiente que realiza una funcionalidad específica y tiene su propio gestor de eventos, su bucle principal y su motor gráfico para mostrar la información por pantalla. (3)

Los estados permiten separar ciertas tareas que son necesarias para la puesta a punto del juego.La subdivisión en estados depende de cómo se vaya a estructurar el juego. No existen reglas en las que basarse para decidir cuándo es conveniente separar dos tareas en dos estados diferentes, pero principalmente dependerá de si estás pueden compartir gestores de eventos y los diferentes motores del juego, o no.

Por lo general en un videojuego los estados más comunes son:

- Inicio.
- Menú principal.
- Bucle principal.
- Pausa.
- Salir

Dentro de cada uno de los estados se encuentra un bucle que se ejecuta continuamente. El estado que esté activo indicará si se está en el menú principal esperando que el usuario seleccione como quiere jugar o bien en el estado "de juego", donde transcurre la acción de la partida.

Cada estado tiene un gestor de eventos propio que se encarga de procesar todos los eventos que estén permitidos mientras se esté en el estado y además tiene que renderizar la pantalla para mostrar la información necesaria con la que el usuario pueda interactuar. En el caso del menú puede bastar con un simple *engine* 2D que sea capaz de mostrar las opciones al usuario, en cambio, cuando se esté en el bucle principal del juego posiblemente se necesite usar un *engine* gráfico mucho más complejo, de tipo 3D

1.4.2. Diagrama de estados. Mapas de navegación

Un diagrama (o máquina) de estados finitos define una serie de estados y una relación entre ellos. De todos los estados, sólo uno de ellos está activo y, a partir de una serie de entradas, se realizan transiciones entre estados y cada cambio genera una serie de salidas. (3)

Las características básicas de este tipo de estructuras son:

- Se pueden implementar con un código muy sencillo y limpio.
- Son estructuras fáciles de depurar.
- El código de control es mínimo y no generan "*overhead*".
- Son muy predictivas.

Las máquinas de estado finito pueden representarse mediante grafos y estos a su vez mediante mapas de navegación, que no son más que una representación esquemática que muestra la estructura, en un espacio, de determinados elementos y la interrelación entre ellos.

En el caso específico de los estados de un videojuego, cada componente corresponde con un estado y las aristas definen las transiciones entre estados. En las aristas se incluye información como, por ejemplo,

condiciones que deben cumplirse. Mediante un mapa de navegación se puede visualizar las posibles transiciones que existen entre los diferentes estados del juego.

La Fig.2 esquematiza un ejemplo de un posible mapa de navegación para un videojuego.

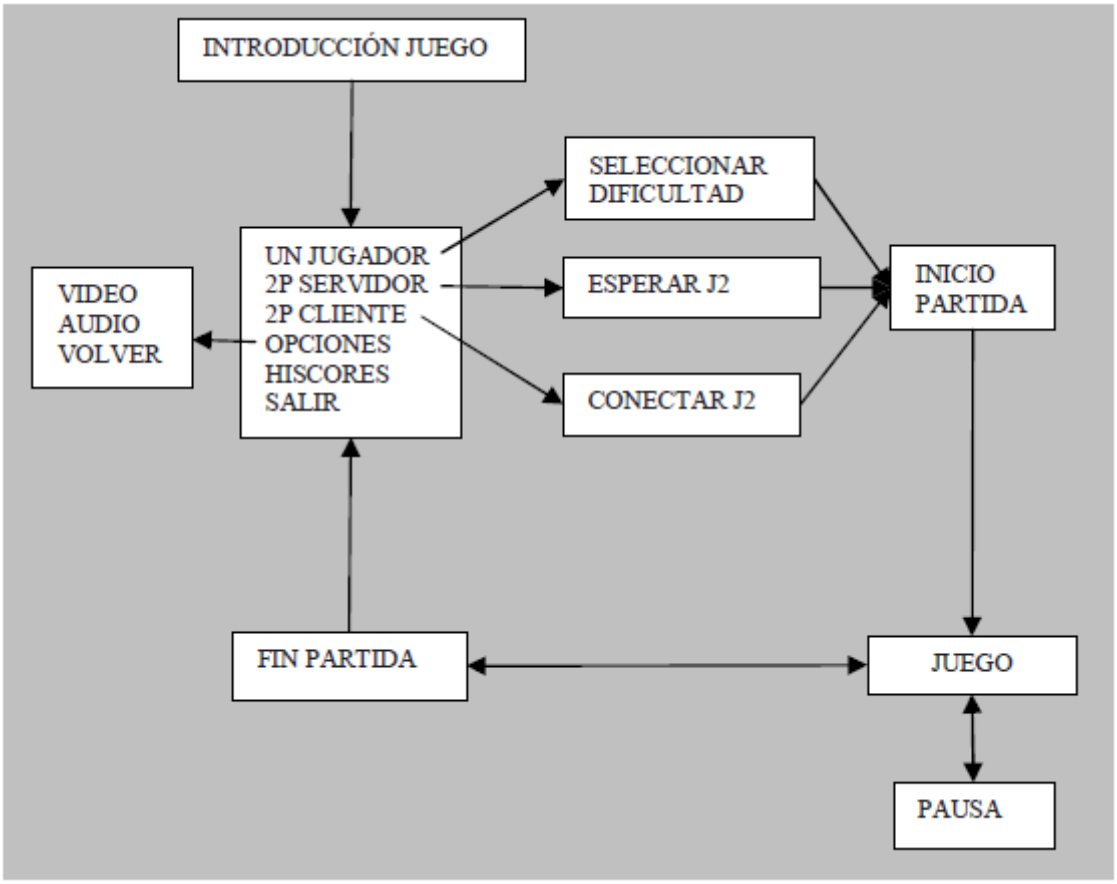


Fig. 2 Mapa de navegación

A continuación se brinda su descripción:

- Después de mostrar algunos videos introductorios en un primer estado, el estado que se carga es el del menú, donde se tratan sólo aquellos eventos que permiten seleccionar que se quiere hacer en el juego: si jugar solos, crear un servidor de juegos, conectarse a un servidor, modificar las opciones de juego o salir.
- En las pantallas del menú se configura el comportamiento que tendrá el juego. Se puede modelar cada una de estas opciones dentro de un estado propio o ponerlas en un estado más genérico que incluya toda la configuración del juego.
- Una vez el usuario haya seleccionado un tipo de juego, se puede comenzar la partida mostrando una animación o video (para posicionar los personajes, etc.), y finalmente se pasa el control al estado que permite jugar al usuario. Allí todo gira alrededor del bucle principal que es donde se encuentra implementado el auténtico juego.
- Una vez el juego finalice, se devuelve el control al estado del menú. (3)

1.3.1. Implementación del gestor.

La forma tradicional de implementar un sistema de estados ha sido siempre utilizando comandos bucles *if* y *case*. Solo se tiene un programa principal, y en cada paso del bucle se busca el estado en el que se está y allí se ejecuta el código para tratarlo.

El problema de esta implementación es que no permite tener varios motores gráficos o gestores para cada uno de los estados del juego. Además, cada vez que se recibe una entrada de teclado o de red se tiene que verificar si esto provoca un cambio de estado, con lo que es más fácil tener incoherencias en el sistema. Todo esto hace que el programa principal sea difícil de seguir y, por tanto, difícil de mantener y depurar.

Otra alternativa más eficiente es utilizar la ingeniería del software para diseñar e implementar un gestor de estados que permita modular el comportamiento de los mismos. Un gestor de estados se compone de dos partes:

- Cada estado es una clase que implementa una serie de funciones comunes: iniciar el estado, finalizar el estado, actualizar la pantalla, gestionar determinados eventos.
- Se tiene un controlador de estados que se encarga de indicar cuál es el estado que se encuentra activo en todo momento y permite cambiar a otro estado.

Para coordinar todos los estados, se debe crear un controlador de estados, al que se puede nombrar *StateManager*. El *StateManager* debe proporcionar funciones para seleccionar el estado actual o guardar un estado temporalmente (por ejemplo si se hace una pausa). (3)

1.3.2. El bucle principal del "estado jugar"

Se analizará la estructura interna del bucle principal del estado "de jugar", para mostrar cómo se integra la información que se recibe y envía a todos los componentes que se utilizan durante el juego, básicamente: entrada, sonido, gráficos y red. (3)

El bucle principal del juego es la parte que se ejecuta continuamente y es donde se produce todo el desarrollo de la partida.

El bucle se puede ejecutar de dos formas posibles:

- Se puede ejecutar el bucle cada vez que se genera un evento especial, como por ejemplo cada vez que se tiene que refrescar la pantalla.
- Otra opción es estar ejecutándolo todo el tiempo utilizando alguna técnica multihilo.

El bucle principal está compuesto por tres fases principales: la recogida de datos, el cálculo del nuevo estado global del juego y la redistribución de los cambios.

La implementación de este bucle principal se debe realizar íntegramente dentro de cada estado donde tenga lugar el desarrollo de la partida. En otros estados se pueden eliminar algunos de estos puntos.

De las tres fases por la que está compuesto el bucle principal del estado jugar se centra en el cálculo del nuevo estado global del juego.

Una vez que el videojuego ya en el estado jugar haya realizado todas sus acciones y movido todos sus elementos es necesario recalcular el estado del sistema y actualizar las variables globales. Este paso se realiza al final de todas las acciones para poder integrar el resultado de las acciones colectivas de los elementos que han actuado en esta iteración. Algunos de las tareas que se deben realizar en este paso final son:

- Es necesario incrementar el tiempo del juego proporcionalmente. Por ejemplo, hay juegos donde es importante la distinción entre noche y día.
- En segundo lugar, se tiene que mirar si el juego ha llegado a su fin. En este caso se informa al bucle principal que hay que cambiar de estado para dejar de procesar las entradas del usuario.
- También se puede mirar si se ha cumplido algún objetivo intermedio y, por lo tanto, si es necesario cambiar.

Toda esta información se deja almacenada en las estructuras de datos del sistema y se avisa a los componentes que lo necesiten (IA, gráficos,…) de todos los cambios que se han producido para que se puedan actualizar con el nuevo estado del juego. (3)

1.4 Motor de interfaz

El motor de Interfaz es la parte encargada de interactuar directamente con el jugador, y mantener el diálogo entre éste y el juego. Se encarga de presentar todos los contenidos, opciones, escenas del mundo virtual, y también de los controles necesarios para poder interactuar dentro del videojuego, así como mostrarnos el *"look & feel*" [2] final de éste. Es un elemento muy importante, pues es lo primero que

[2] Estilo visual del videojuego.

conocerá el usuario del videojuego y debe ser lo más sencillo y familiar posible a la naturaleza del juego, ya que influye en gran medida en la jugabilidad final del juego. (4)

Tras haber señalado el concepto de motor de interfaz es conveniente, para la total comprensión del mismo, conocer el concepto de interfaz y más específicamente de interfaz gráfica de usuario.

1.4.1. Interfaz gráfica de usuario

La creación de software va generalmente unida al binomio: juegos de ordenador e interfaz visual. Si se analiza etimológicamente la palabra interfaz se aprecia que es una palabra compuesta por dos vocablos: Inter proveniente del latín inter, y cuyo significado es “entre” o “en medio”; y el vocablo Faz el cual proviene del latín facĭes, y tiene como significado “superficie, vista o lado de una cosa”. (4)

Por lo que la idea fundamental en el concepto de interfaz es el de mediación, entre hombre y máquina. La interfaz es lo que "media", lo que facilita la comunicación, la interacción, entre dos sistemas de diferente naturaleza, típicamente el ser humano y una máquina como el ordenador. Esto implica, además, que se trata de un sistema de traducción, ya que los dos "hablan" lenguajes diferentes: verbo-icónico en el caso del hombre y binario en el caso del procesador electrónico.

De una manera más técnica se define a Interfaz de usuario, como conjunto de componentes empleados por los usuarios para comunicarse con las computadoras. El usuario dirige el funcionamiento de la máquina mediante instrucciones, denominadas genéricamente entradas. Las entradas se introducen mediante diversos dispositivos, por ejemplo un teclado, y se convierten en señales electrónicas que pueden ser procesadas por la computadora. (5)

Cuando se habla de interfaz gráfica de usuario, el concepto es aún más específico ya que la interfaz gráfica de usuario al contrario del concepto de interfaz tiene una localización determinada y definida: Si la interfaz etimológicamente supone la cara o superficie mediadora, la interfaz gráfica de usuario, supone un tipo específico de interfaz que usa metáforas visuales y signos gráficos como paradigma interactivo entre la persona y el ordenador.

Se podría definir la interfaz gráfica de usuario, en el contexto de la interacción persona-ordenador, como un artefacto interactivo, que por su diseño, posibilita la interacción de una persona con el sistema

informático, haciendo uso de las gramáticas visuales y verbales (signos gráficos como iconos, botones, menús y verbales como tipografía). (5)

1.4.2. Teoría del diseño de interfaces gráficas en los videojuegos

El diseño de interfaces es una disciplina que estudia y trata de poner en práctica procesos orientados a construir la interfaz más usable posible, dadas ciertas condiciones de entorno. Pertenece a un campo mayor del conocimiento humano, de origen altamente interdisciplinario, llamado *Human Computer Interaction.* La importancia del diseño de una buena interfaz se sintetiza en la siguiente figura. (6)

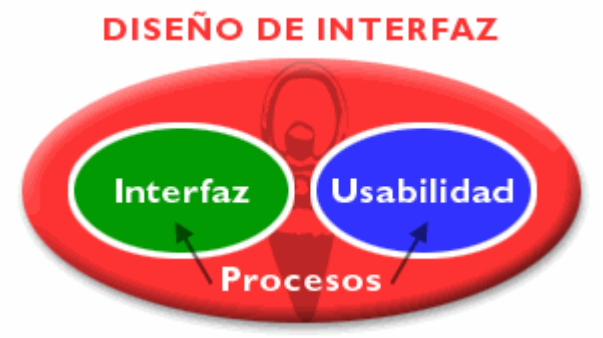


Fig. 3 Importancia del diseño de interfaz

La finalidad primordial de un videojuego debe ser la de recrear al usuario, logrando ser capaz de captar la atención del mismo. Existen varios aspectos que conciernen en ese sentido. En primer lugar, un videojuego debe de crear un ambiente que incite el interés en el jugador, en este punto es donde la presentación y la ambientación del videojuego entran a desempeñar un rol protagónico por dos razones fundamentales:

- Al usuario debe serle posible identificar todos los aspectos del videojuego, a nivel visual y auditivo, para poder llevar a cabo sus acciones.

- La ambientación es el factor principal para que el usuario se sienta identificado con el personaje y la historia presentados.

Por lo tanto, teniendo en cuenta la relación que existe entre interfaz y presentación, se deriva la importancia de la interfaz gráfica de usuario (GUI) en un videojuego, pues es el artefacto tecnológico que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable del usuario con el juego. Una buena interfaz lo tornará más divertido, de rápido uso y de fácil comprensión, factores que establecen buenas prácticas para los que incursionan en el diseño y creación de videojuegos.

Los juegos llevan décadas ofreciéndonos metáforas visuales, interfaces de muchos tipos y diseño de entornos textuales, bidimensionales o tridimensionales, siempre altamente inmersivos. Desde los primeros *MODS*[3] hasta los *SIMS*[4], pasando por *Mario Bross*, el *Tetris*, el *DOOM* y *Age of Empires*. El desarrollo de una interfaz de un videojuego requiere algunas características especiales que lo diferencian de una interfaz para cualquier otra aplicación. El uso correcto de la interfaz de usuario junto con el diseño de un buen sistema de entrada son claves para facilitar que el usuario aprenda deprisa el funcionamiento del juego. Por el contrario, una mala interfaz puede provocar frustración en el jugador.

Es muy importante que al trabajar en el diseño de una interfaz de usuario, la prioridad sea ofrecer toda la información necesaria para el usuario de una forma eficiente, en lugar de centrarnos principalmente en que la información se presente de forma bonita o no. La información que es necesaria mostrar depende del tipo de juego que se desarrolle (por ejemplo, en un juego de estrategia es abundante, en cambio, en uno de aventura gráfica, es mínima), así que se debe estudiar cada caso en particular.

El videojuego que se pretende hacer se debe diseñar pensando en cómo va a jugar el usuario final. Para ello, primero hay que estudiar el perfil del usuario al que va dirigido nuestro producto y las capacidades de

[3] Es una extensión que modifica un juego original proporcionando nuevas posibilidades, ambientaciones, personajes, diálogos, objetos, etc.

[4] Videojuego de simulación social y estrategia.

hardware de la plataforma sobre la que se desarrollará; a partir de este estudio, habrá que definir el diseño de la aplicación. Hay que tener en cuenta que modificar un juego que ya está diseñado para adaptarlo a un tipo de usuario concreto es una tarea muy complicada y se debe evitar en lo posible.

Una medida para conocer el grado de relación existente entre un usuario y la aplicación es lo que se conoce por usabilidad. Un sistema con una buena usabilidad nos permitirá:

- Reducir el tiempo que se necesita para hacer una tarea.
- Reducir el número de errores que se cometen al interactuar con el sistema.
- Reducir el tiempo que se necesita para aprender a interactuar con el sistema.

Algunas recomendaciones genéricas que permitirán mejorar la interfaz de usuario de cualquier juego se exponen a continuación:

- Es importante conocer qué decisiones puede tomar el jugador en todo momento, y se deben presentar en la pantalla toda la información posible para que el usuario pueda decidir cuál es su próximo movimiento. Toda la información relevante se debe mostrar de forma simultánea.
- La información se debe agrupar coherentemente para que el jugador, con una mirada rápida, pueda captar el máximo de información. Una buena estructura en grupos de contenidos se puede ayudar a presentar mucha más información sin llegar a colapsar al jugador.
- El tiempo de respuesta de la interfaz de usuario tiene que ser mínimo y estar al 100% sincronizado con todos los eventos que suceden en el juego, es decir, no se puede refrescar la interfaz de forma independiente del resto del programa.
- Entre dos diseños de interfaz que se proporcionen acceso a la misma información, siempre elegir el que lo haga de forma más simple e intuitiva. (7)

1.4.3. Buenas prácticas en el diseño de interfaces gráficas en los videojuegos

- ✓ **Metáfora**

No se puede hablar de diseño de las interfaces gráficas, sin aludir a los conceptos de metáfora, modelo mental y modelo conceptual.

La metáfora es la aplicación de una expresión a un objeto o concepto para explicar su semejanza con otro.

En el diseño de interfaces también se usa el concepto de metáfora para definir comportamientos de elementos que componen la aplicación. La metáfora más conocida es la del escritorio de usuario, la cual fue introducida por Xerox y popularizada por Apple. Consiste en definir la ventana principal de un ordenador como un escritorio donde se tiene documentos guardados en carpetas y estos se pueden editar, copiar, pegar y mover.

Al interactuar con cualquier elemento de la aplicación, se debe elegir su comportamiento para que sea fácil de usar y requiera un periodo de aprendizaje corto.

En la mayoría de los casos es muy recomendable utilizar metáforas con las que ya se trabaja y que, gracias a su popularidad, permitirán que el usuario sólo necesite adaptar su conocimiento al nuevo uso en la aplicación. (8)

- ✓ **Dónde utilizar las metáforas**

Las metáforas se utilizan en todos los aspectos de interacción de la aplicación y se debe ser coherente con su uso. El usuario encontrará las metáforas en los siguientes puntos:

- Elementos para la configuración del juego (número de jugadores, nivel del juego, teclas para jugar, volumen del juego).
- Elementos del juego: cartas, armas, atrezo.

En una aplicación como puede ser un videojuego, se tiene la licencia de trabajar con interfaces más artísticas y deberíamos aprovecharla. (9)

- ✓ **Ejemplos de metáforas**

En el caso de los videojuegos, se pueden encontrar varias metáforas. Ya de por sí, un escenario 3D es una metáfora visual de un mundo real. Las formas de interactuar con éste, también se pueden considerar una metáfora.

Los menús de las aplicaciones también son otra metáfora que representan todas las opciones a disposición del usuario. Se pueden realizar mediante el uso de cadenas de texto o mediante una forma mucho más visual.

- ✓ **Modelos**

Según la psicología, el ser humano almacena el conocimiento en estructuras semánticas. Estas estructuras se organizan a partir de la interacción con el mundo externo usando los procesos perceptuales. Esta información se almacena en diferentes tipos de memoria y se rechaza o se mantiene según diversos criterios, lo que se puede llamar "aprendizaje".

Existen dos modelos que permiten estudiar el aprendizaje de un determinado concepto por parte de una persona:

- **Modelo mental**. A partir de las experiencias y el aprendizaje realizado por la persona, éste tiene un modelo inexacto y sólo conoce relaciones sencillas del concepto estudiado ("si se cumple esta condición, ocurre esto").Por ejemplo una persona, sabe que si presiona el acelerador de un coche, éste se mueve, pero no sabe el por qué (no entiende de mecánica).
- **Modelo conceptual**. El estudio del concepto se hace de manera detallada y se conocen todos los mecanismos que actúan sobre él. Es el modelo detallado de un concepto. Siguiendo con el ejemplo anterior, un ingeniero mecánico conoce al detalle cómo está construido un coche. (10)

En el caso de la programación de aplicaciones, el modelo que tiene el programador sobre su aplicación sería un modelo conceptual, mientras que el modelo que se forman los diferentes usuarios sería un modelo mental.

Para una misma aplicación, los dos modelos deben asemejarse tanto como sea posible. Cuanta más divergencia haya entre ellos, más probable será el rechazo por parte del usuario. Si se desea que el

videojuego sea intuitivo, se debe pensar en cómo crear un modelo conceptual que se asemeje lo mayormente posible al del usuario.

1.4.4. Diseño de interfaces gráficas en los videojuegos

Los componentes básicos de una interfaz gráfica son:

- Etiqueta.
- Cuadro de texto.
- Cuadro de confirmación.
- Grupo de selección.
- Barra de desplazamiento.
- Lista de elementos.
- Botón

Se pueden utilizar diferentes bibliotecas que implementen esta serie de componentes. En el caso de un videojuego, estos elementos deben de ser muy personalizables para que se integren bien en el diseño global de la aplicación. (11)

✓ **Etiqueta**

Una etiqueta es una representación de un texto sobre la ventana de juego.

✓ **Cuadro de texto**

Este elemento gráfico presenta un texto que puede modificar el usuario. En entornos estándares suele utilizarse el teclado para introducir texto, pero cuando se trabaja con videoconsolas, este periférico no suele estar disponible.

✓ **Cuadro de confirmación o checkbox**

Estos elementos suelen tener dos posibles estados y suelen simbolizarse con un cuadrado vacío que puede contener un aspa o un indicador para diferenciarlos. En entornos artísticos se tiene la licencia de

modificar este comportamiento, pero debe considerarse la simplicidad y representar correctamente la información al usuario. (12)

Un error que se suele cometer con este tipo de elementos es la dificultad de entender los dos estados. Por ejemplo, en un cuadro que sólo cambia de color (rojo y verde), no siempre podemos reconocer qué indica cuando está seleccionado.

Normalmente, este elemento está acompañado de una etiqueta que indica qué significa esta selección.

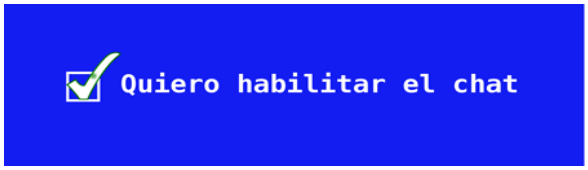


Fig. 4 Cuadro de confirmación

✓ **Grupo de selección**

Este elemento es una agrupación de cuadros de chequeo donde se aplica la restricción de que sólo uno puede estar seleccionado.

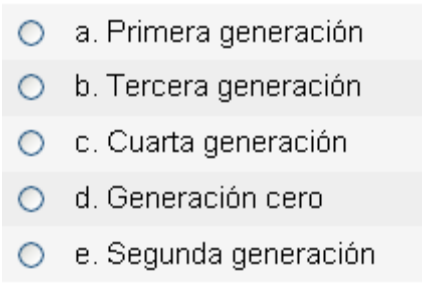


Fig. 5 Grupo de selección

✓ **Lista de elementos**

Este componente presentará al usuario una serie de elementos y éste deberá seleccionar uno (o varios) de ellos. En el caso concreto de un videojuego, esta información se puede presentar de forma más creativa. Algunas posibilidades son:

- Posicionar los elementos (texto, dibujo, animación...) en vertical y resaltar el elemento actual. A partir del cursor, modificar la selección arriba o abajo hasta pulsar con un botón y seleccionar el elemento actual.
- En el caso de que los elementos no quepan en la pantalla, se puede hacer un desplazamiento de los elementos para ir presentándolos todos. Es posible que el elemento que está seleccionado esté siempre centrado en la pantalla.
- Posicionar los elementos en horizontal y trabajar con ellos de manera análoga a la técnica anterior.

Fig. 6 Lista de elementos

La primera técnica presenta mucha información en la pantalla. Por un lado, permite tener una idea global de lo que se está seleccionando, aunque usualmente no se puede enriquecer con información adicional.

En la segunda técnica ocurre lo contrario. Mientras el usuario está seleccionando un elemento, apenas se pueden presentar otras opciones, pero se tiene mucho espacio en la pantalla para presentar información adicional al elemento seleccionado. (13)

✓ **Barra de desplazamiento**

Otro elemento que se utiliza en el diseño de interfaces es la barra de desplazamiento. Su metáfora se basa en las barras de desplazamiento que se han utilizado en mecánica y electrónica durante varios años (ecualizadores de equipos de música, controladores de volumen, controladores de presión en las máquinas de escribir).

Es una excelente manera de ajustar los valores que abarcan una amplia gama de posibilidades. Si el valor que se ajusta sólo tiene un pequeño número de opciones, como fácil, medio, y difícil, entonces se deben adoptar otros métodos de trabajo. Rangos amplios de valores numéricos como del 1 a 10 ó del 0 a 100 son ideales para el uso de barras de desplazamiento.

Consisten en un recorrido normalmente recto o circular que simboliza un número escalar (de cero a cien). En este recorrido existe un apuntador que puede moverse por el trazado y sirve para seleccionar un valor determinado. Este control suele venir acompañado de una etiqueta de texto que representa el valor seleccionado, como por ejemplo un porcentaje. (14)

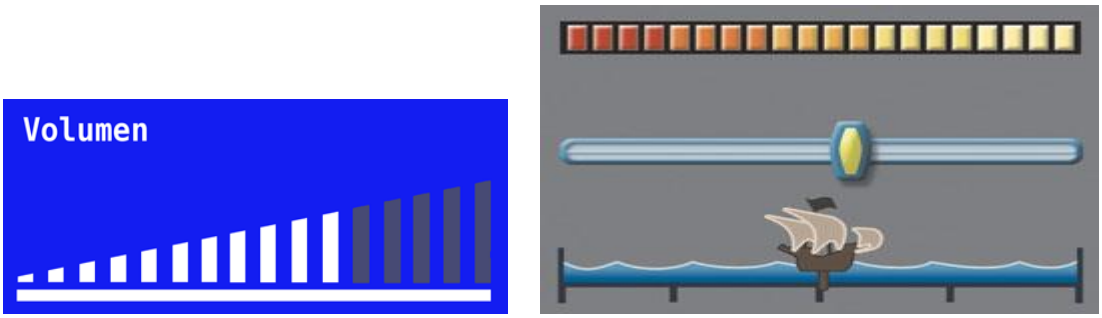


Fig. 7 Opciones para representar una barra de desplazamiento

✓ **Botón**

Metáfora común utilizada en interfaces gráficas con objetivo similar al de un botón corriente. Suele representarse como un rectángulo con una leyenda o icono dentro, generalmente con efecto de relieve. Consta de dos imágenes que cambian en dependencia del estado del botón (oprimido o no oprimido). El usuario puede pulsarlo para efectuar acciones indicadas por él.

Las mejores interfaces tienen un estilo muy distintivo porque capturan el sentido del juego. Estas interfaces no ocurren por accidente. A menudo, son el resultado de un trabajo muy duro. La interfaz es un

componente vital en el juego por lo que no debe ser tratada como un módulo que requiera menos atención, tiempo y talento.

1.4.4. Tendencias del futuro

El estado emocional o físico del jugador no se considera habitualmente parte de la interfaz del juego. En ciertos juegos experimentales que utilizan tecnologías de Computación Afectiva es posible recibir realimentación biológica del jugador para identificar las emociones que siente un jugador en cada momento del juego. Por ejemplo la conductividad eléctrica de su piel puede dar una idea del estrés a que está sometido el jugador y esta información puede utilizarse para que el juego se vuelva más fácil o difícil en función del autocontrol y la calma del jugador.

Muchos de estos prototipos experimentales han sido comercializados ya. Hoy día las consolas de última generación incorporan dispositivos para capturar el movimiento o la imagen de un jugador (como por ejemplo una cámara, una alfombra de baile u otros periféricos similares). Se entiende que estos y otros dispositivos más avanzados formarán parte del interfaz estándar de los videojuegos del futuro, sin embargo, no dejan de ser posibilidades para ir incorporando elementos de la realidad en el juego. (15)

1.5 El videojuego en la Universidad de las Ciencias Informáticas

En la UCI, específicamente en la facultad 5 se han desarrollado videojuegos que ilustran la teoría explicada anteriormente como por ejemplo: “Rápido y curioso” ó “Energía para Aprender”. Este último posee un módulo de presentación con una interfaz adecuada para el tipo de usuario al que va dirigido y acorde el objetivo que persigue. Se puede apreciar la puesta en práctica de los estados de juego lo que no con un diseño de clases lo más eficiente y genérico posible. Para su implementación en su primera versión, se usó el entorno de desarrollo integrado (IDE) Microsoft Visual Studio.Net 2003 y como lenguaje de programación C++ integrado a la herramienta para entornos virtuales SceneTool Kit (STK), también desarrollada en la facultad.

1.6 Motor de juego CNeuro Game Engine. Aplicación en el videojuego Meteorix

El motor de juego Cneuro Game Engine es un framework creado con la finalidad de realizar videojuegos de tipo terapéutico .En estos momentos se le da vida a Meteorix, un videojuego con gran impacto social, analizado desde el punto de vista de la Ludología[5], para el tratamiento de niños con afecciones de ambliopía.

Este motor dentro de sus funcionalidades principales tiene, gestionar niveles del juego y perfiles del usuario, mover los jugadores por el entorno, generar entornos en tiempo de ejecución, utilizar un algoritmo adaptativo que va interactuando constantemente con el juego, registrando el avance del jugador y proporcionando nuevas tareas hasta que se cumplan los objetivos determinados del juego. El motor cuenta con varios módulos, para la visualización el motor gráfico *SceneTool Kit* (STK), para el sonido la biblioteca *SoundToolKit* y para la simulación dinámica y la detección de colisiones usa ODE, pero para lograr un motor de videojuego más completo se requiere de un módulo que gestione las funciones explicadas en teoría a lo largo del capítulo.

[5] Se ocupa del análisis del juego desde la perspectiva de las ciencias sociales, la informática y las humanidades.

Capítulo 2: Características del sistema.

1.1. Introducción

Este capítulo es un profundo acercamiento a las características del módulo de estados de juego y componentes de presentación a desarrollar. Se concretan sus funcionalidades a través del análisis y captura de sus requisitos funcionales y no funcionales y se obtienen los casos de usos del sistema que guiarán el diseño e implementación del módulo. Además, se realiza una breve alusión a la metodología de desarrollo de *software*, lenguaje de modelado y herramienta CASE que se utiliza así como al entorno de desarrollo integrado (IDE) y al lenguaje de programación que se usa para llevar a vía de hechos la solución.

2.2. Objeto de automatización

Con la presente investigación se pretende automatizar el proceso de navegabilidad y presentación para el motor de videojuego CneuroGameEngine, mediante la creación de un módulo de estados de juego y componentes de presentación que implemente un gestor de estados de juego y las distintas ventanas que componen la interfaz visual del videojuego.

2.3. Información que se maneja

Se manipula la información referente a la lógica del funcionamiento de los estados de juego y los estándares de máquinas de estados de juego que generalmente se aplican en los videojuegos. Se opera además con las características del CneuroGameEngine en términos de funcionamiento y estructuración, factor sin el cual no se podría determinar cómo subdividir las funcionalidades del motor en estados de juego.

2.4. Propuesta del sistema

Para solventar la situación problémica existente se propone el desarrollo de un módulo de estados de juego y componentes de presentación, el cual estará compuesto por un gestor de estados que posibilitará la división del videojuego en estados de juego y permitirá la navegabilidad entre dichos estados y a través del videojuego.

Se compondrá además por las distintas ventanas de la interfaz visual del juego: la ventana principal donde se carga el menú del juego que posibilita al usuario la ejecución de las acciones que controlan el funcionamiento del mismo, la ventana opciones donde el usuario configurará las teclas a usar durante el desarrollo de la partida, el nivel del volumen o si desea usar pantalla completa o no y la ventana que permitirá crear el perfil de jugador con la inserción por el usuario de los datos requeridos que consecutivamente serán almacenados en un fichero con el fin de reutilizarlos.

El gestor de estados estará compuesto por cinco estados: inicio, menú, jugar, pausa, salir y además un administrador de estados que coordinará en qué momento debe activarse cada uno

En el estado inicio se cargará un video introductorio mostrando la historia del videojuego para insertar al usuario en el ambiente del mismo.

En el estado menú se gestionará la carga e inicialización del menú principal del juego lo que le permitirá al jugador navegar hacia los distintos estados de juego, lógicamente este proceso le es transparente al usuario. Agregando a lo anterior la gestión de la ventana opciones y la de perfil de jugador.

En el estado jugar, el más importante en términos funcionales, se la da inicio al bucle principal que realiza las acciones que rigen la dinámica del juego. Se cargan todos los recursos para poder dar inicio a la partida: la escena, los niveles de juego, el generador de entornos y el cañón.

El estado pausa efectuará la función de pausar el juego mostrando una ventana de confirmación al usuario de si desea salir del juego o no .En caso afirmativo se retorna al estado menú y en caso negativo se continúa con la partida.

En el estado salir se carga un video final y se cierra la aplicación.

El módulo se integrará al motor de videojuego Cneuro Game Engine, el cual posee un patrón arquitectural basado en tres capas: la de presentación, la lógica y la de soporte. Los elementos del módulo a desarrollar que gestionan lo referente a la interfaz gráfica, componentes visuales y de presentación se encuentran en la capa de presentación del motor y los que se encargan de los estados de juego están ubicados en la capa lógica.

A continuación se muestra un diagrama que esquematiza lo explicado anteriormente.

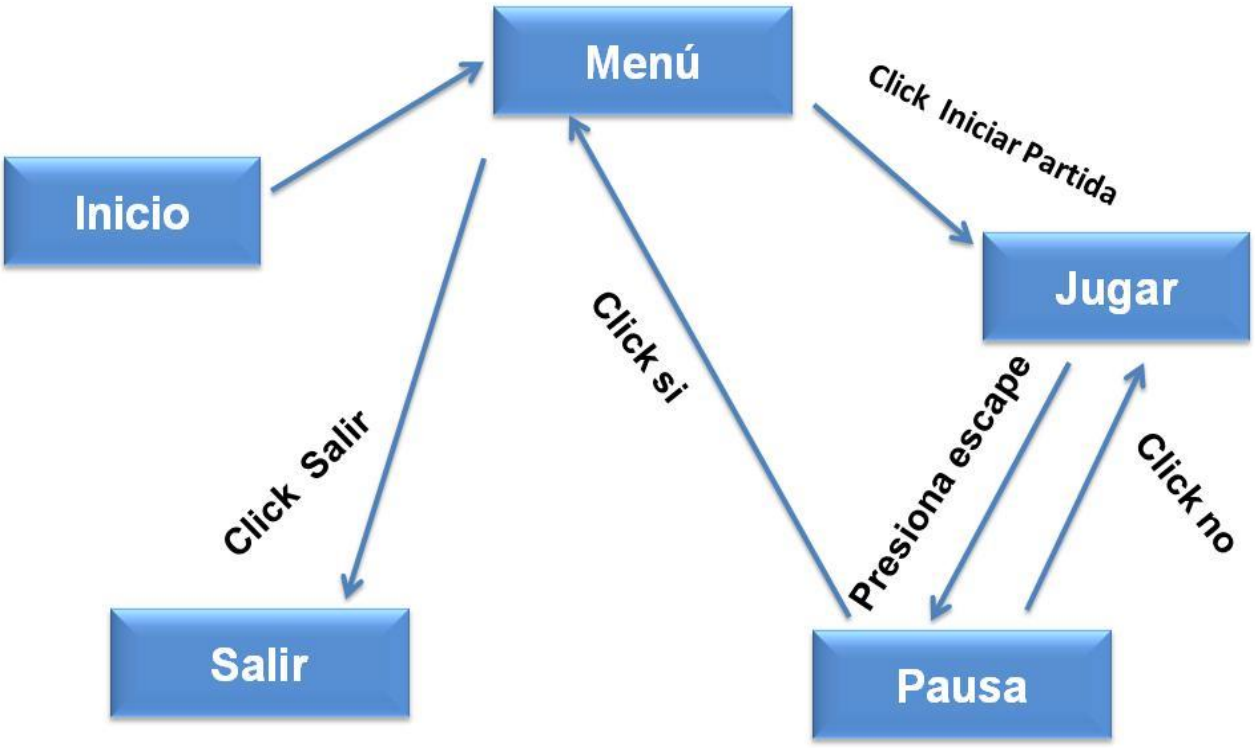


Fig. 8 Estados de juego del sistema

2.5. Metodologías, herramientas y lenguajes de programación y modelado.

Para la implementación del módulo a desarrollar se hará uso del entorno de desarrollo integrado (IDE) Microsoft Visual Studio.Net 2003 debido a que la biblioteca para entornos virtuales *Scene Tool kit* (STK) que se utilizará, lo requiere para su funcionamiento.

El lenguaje de programación a utilizar será el C++ porque posee un sinnúmero de características que le dan ventaja sobre otros lenguajes para su uso en el campo del desarrollo de videojuegos complejos y en tres dimensiones (3D). Una de las razones de programar en C++ es su increíble versatilidad y flexibilidad. Es un lenguaje multi-nivel y multi-paradigma, se puede usar tanto para programar directamente el hardware (dependiendo del sistema operativo), como para crear aplicaciones tipo Windows definidas todas por poseer una misma interfaz. Brinda soporte para diversas librerías así como dispone de sus propias librerías que implementan varias estructuras de datos de forma genérica. (16)

La metodología de desarrollo de software que guiará la construcción del sistema será *Rational Unified Process* (RUP) junto al lenguaje Unificado de modelado UML, por constituir una metodología robusta, la más estándar y utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

La herramienta CASE seleccionada para el desarrollo del proceso ingenieril es Visual Paradigm por ser considerada como una de las más completas, de fácil uso para el modelado y presentar características gráficas muy cómodas para el usuario. Sumándoles a dichas ventajas que es multiplataforma.

2.6. Modelo del dominio

El Modelo de Dominio (o Modelo Conceptual) es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema. Representa conceptos del mundo real, no de los componentes de software, además es importante señalar que se le considera en RUP un subconjunto del llamado modelo de objetos del negocio.

Este modelo se realiza a través de un diagrama de clases de UML simplificado, en el cual se representan las clases conceptuales que pueden intervenir en el sistema y sus asociaciones preliminares, así como los objetos más importantes en el mismo.

Estos objetos del dominio representan "cosas" que existen o los eventos que acontecen en el medio en el que se desenvuelve la aplicación. (17)

2.6.1. Diagrama del modelo del dominio

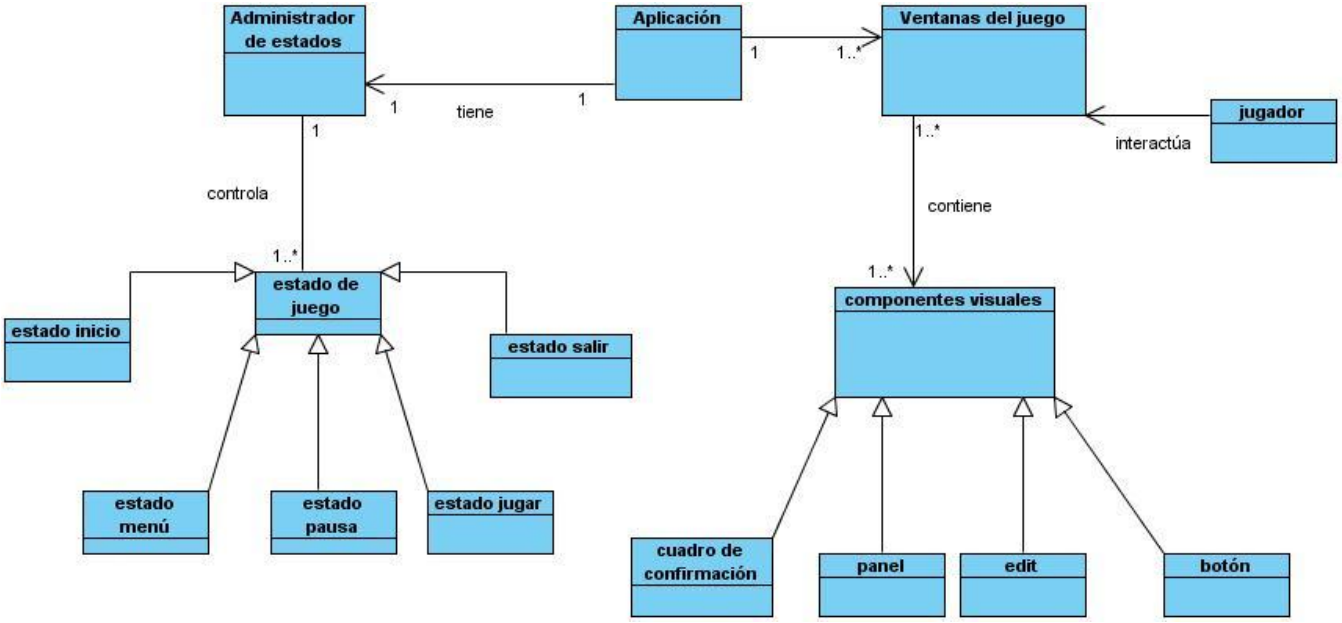


Fig.9 Modelo del dominio

2.6.2. Glosario de Términos del dominio

- **Administrador de estados:** Coordina los estados del videojuego y las transiciones entre ellos.
- **Aplicación:** Se encarga de ejecutar las funcionalidades del módulo.

- **Componente de presentación:** Elemento que compone la presentación del videojuego. Se encarga de presentar todos los contenidos, opciones, y controles necesarios para que el jugador pueda interactuar dentro del videojuego.
- **Ventana de juego:** Componente de presentación**.** Interactúa directamente con el jugador, y mantiene el diálogo entre éste y el juego.
- **Componente visual:** Elemento que forma al componente de presentación. Mínima unidad de la presentación del videojuego.
- **Botón:** Componente visual. Metáfora común utilizada en interfaces gráficas con objetivo similar al de un botón corriente. Suelen representarse como un rectángulo con una leyenda o icono dentro, generalmente con efecto de relieve. Consta de dos imágenes que cambian en dependencia del estado del botón (oprimido o no oprimido). El usuario puede pulsarlo para efectuar acciones indicadas por él.
- **Panel:** Componente visual. Muestra una imagen de fondo en las ventanas de juego.
- **Cuadro de confirmación:** Componente visual. Este elemento tiene dos posibles estados y se simboliza con un cuadrado vacío que contiene un aspa o un indicador para diferenciarlos.
- **Estado de juego:** Módulo independiente que realiza una funcionalidad específica. Separa ciertas tareas que son necesarias para la puesta a punta del videojuego**.**
- **Estado Inicio:** Estado que le da introducción al videojuego**.** Se renderiza un video introductorio .Luego se inicializa el estado menú.
- **Estado Menú:** Estado que controlará los eventos que le posibilitarán al usuario controlar el juego a su comodidad. Se podrá transitar de él hacia el estado jugar o hacia la salida de la aplicación.
- **Estado Pausa:** Posibilita la suspensión de las tareas del juego en un intervalo de tiempo determinado, reanudándolas nuevamente cuando se desee.
- **Estado Jugar**: Es el estado con mayor nivel de funcionalidad dentro del videojuego porque empaqueta el bucle principal del juego. De él se puede transitar hacia el estado pausa o hacia el estado menú.
- **Jugador:** Interactúa con el videojuego y se beneficia de él.

2.7. Modelo del sistema

A partir de este momento se comienza el modelado del sistema que se va a desarrollar. Con este fin se realiza la captura de requisitos funcionales y no funcionales y se identifican los fragmentos de funcionalidades del sistema en términos de casos de uso.

2.7.1. Requisitos de Software

Los requerimientos de un sistema definen qué es lo que este debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen. Un requerimiento es una característica de diseño, una propiedad o un comportamiento de un sistema. Estos constituyen la descripción de los deseos o de las necesidades de un producto. Se pueden clasificar en requerimientos funcionales y no funcionales.

- **Requerimientos funcionales**: son capacidades o condiciones que el sistema debe cumplir.

- **Requerimientos no funcionales:** Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto. Normalmente, están vinculados a requisitos funcionales. (18)

Al sistema a desarrollar se le efectúo la captura de los siguientes requisitos:

- **Requisitos funcionales:**

El sistema debe permitir:

RF1- Manipular estados de juego

RF 1.1 Ejecutar estado Inicio

RF 1.2 Ejecutar estado Menú

RF 1.3 Ejecutar estado Pausa

RF 1.4 Ejecutar estado Jugar

RF 1.5 Ejecutar estado Salir

RF 1.6 Ejecutar transiciones entre estados.

RF2- Crear perfil del jugador

RF3- Configurar opciones de juego (pantalla completa, teclas a usar, volumen)

RF4- Iniciar juego

RF5- Salir del juego

- **Requisitos no funcionales:**

  - **Requisitos de software**

    RF1.1- Se debe disponer del sistema operativo Windows 95 o superior.

  - **Restricciones en el diseño y la implementación.**

    RF2.1- El módulo se implementa en el Entorno de Desarrollo Integrado (IDE) Microsoft Visual Studio .Net 2003.

    RF2.2- El módulo tiene que ser desarrollado en el lenguaje de programación C++ integrado al motor gráfico *Scene Tool kit* (STK) para el uso en entornos virtuales, acorde con las restricciones del sistema del que va a formar parte.

  - **Requisitos de apariencia o interfaz externa.**

    RF 3.1 - El menú principal estará alineado hacia la izquierda predominando en él el color rojo claro.

RF 3.2 Se hará uso de los componentes visuales: panel, botón, checkbox y edit.

RF 3.3 Las imágenes de fondo cargadas serán de tipo png prevaleciendo en ellas el color carmelita.

RF 3.4 La fuente empleada será de color blanco.

RF 3.5 La resolución por defecto que tendrá la aplicación será de 800x600.

- **Requisitos de confiabilidad**

RF 4.1 El software deberá poseer un número de fallos mínimos, recuperándose rápidamente ante cualquier situación comprometedora. La entrada de datos del usuario al sistema será validada.

2.7.2. Modelo de casos de uso del sistema

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Por lo tanto, establece un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades (requisitos) que debe cumplir el sistema. (19)

- **Actores del sistema**

En este acápite se identifican los actores del sistema y se expone la justificación de su selección. A su vez se definen los casos de uso del sistema con sus respectivas especificaciones, partiendo de la trazabilidad directa que existe entre requisitos funcionales y casos de usos.

Actores	Justificación

Jugador	Es el encargado de interactuar directamente con el motor de interfaz, ejecutando funcionalidades como: iniciar la partida, modificar la configuración global del juego, crear el perfil de jugador y salir de la aplicación.
CNeuroGameApp	Es el sistema responsable de inicializar el gestor de estados, ejecutando la funcionalidad principal del módulo: manipular estados.

Tabla 1 Descripción de los actores del sistema

- **Paquetes de casos de usos**

El diagrama de casos de uso del sistema está dividido en dos paquetes lógicos. Este empaquetamiento se efectuó, con el objetivo de particionar las funcionalidades de acuerdo con las tareas que debe de cumplir cada caso de uso, es decir, cohesionar los más afines y separar los que poseen responsabilidades diferentes para lograr una modelación del sistema de una forma más organizacional.

Fig.10 Paquetes de casos de usos

El paquete manipular estados agrupa el caso de uso “Manipular estados de juego” que describe el funcionamiento del gestor de estados con sus correspondientes estados de juego y las transiciones entre dichos estados.

El paquete componentes de presentación recoge los casos de uso que describen el comportamiento de la interfaz gráfica del sistema desde el punto de vista del jugador: Iniciar juego, crear perfil de jugador, configurar opciones de juego y salir de la aplicación.

El paquete manipular estados posee una relación de dependencia con el paquete componentes de presentación, puesto que los casos de uso que empaqueta, manejan para su ejecución las funcionalidades del paquete componentes de presentación.

- ✓ **Diagrama de casos de uso del paquete Manipular estados**

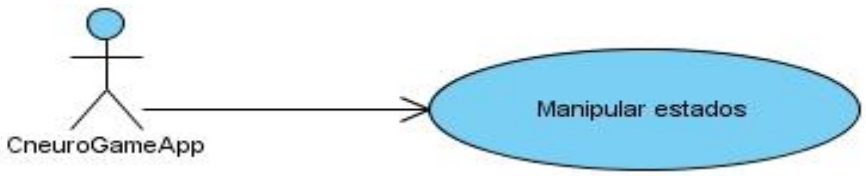


Fig.11 Diagrama de casos de uso manipular estados

- ✓ **Diagrama de casos de uso del paquete componentes de presentación**

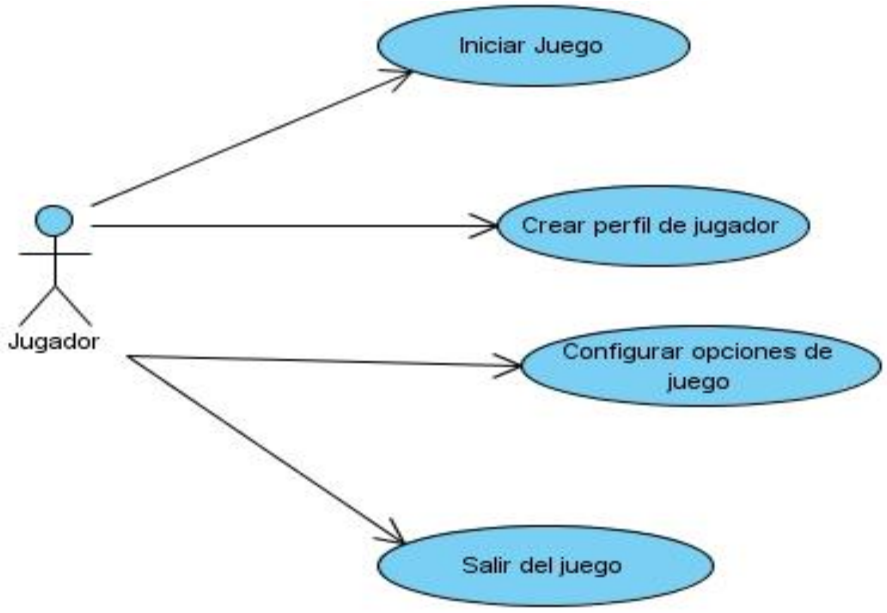


Fig. 12 Diagrama de casos de uso componentes de presentación

2.7.3. Descripción de los casos de uso del sistema

Caso de Uso: 1	Manipular estados de juego
Propósito	Indicar el estado que debe estar activo en la aplicación. Permitir la navegabilidad en el videojuego.
Actores	CneuroGameApp
Resumen:	El caso de uso se inicia desde el mismo instante en que la aplicación comienza a funcionar, se carga el estado menú y se transita luego hacia los demás estados cuando la aplicación así lo requiera, finalizando con el cierre del videojuego.
Precondiciones	La aplicación debe estar corriendo.
Poscondiciones	Se inicia, se finaliza un estado de juego y se ejecutan transiciones entre ellos.

Referencias	RF1
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
1. El CneuroGameApp ordena iniciar el administrador de estados de juegos.	1.1 El sistema puede ejecutar varios estados de juego o transitar de un estado a otro: a) Si se decide ejecutar estado inicio ir a la sección “Ejecutar estado Inicio”. b) Si se decide ejecutar estado menú ir a la sección “Ejecutar estado menú”. c) Si se decide ejecutar estado jugar ir a la sección “Ejecutar estado jugar” d) Si se decide ejecutar estado pausa ir a la sección “Ejecutar estado pausa” e) Si se decide ejecutar el estado salir ir a la sección “Ejecutar estado salir”. f) Si decide ejecutar transiciones entre estados ir a la sección “Ejecutar transiciones entre estados”.
Sección “Ejecutar estado inicio”	
Acción del Actor	**Respuesta del sistema**

2. El CneuroGameApp indica actualizar el administrador de estados.	2.1 El sistema carga un video introductorio que muestra la historia del videojuego. 2.2 El sistema finaliza el estado.
Sección "Ejecutar estado menú"	
Acción del Actor	**Respuesta del sistema**
3. El CneuroGameApp indica actualizar el administrador de estados.	3.1 El administrador de estados inicializa el estado menú. 3.2 El sistema carga todos los componentes del menú principal y los muestra en la ventana principal del videojuego.
Sección "Ejecutar estado jugar"	
Acción del Actor	**Respuesta del sistema**
4. El CneuroGameApp indica actualizar el administrador de estados.	4.1 El administrador de estados inicializa el estado jugar. 4.2 El sistema carga los recursos necesarios y comienza la ejecución del bucle principal del juego
Sección "Ejecutar estado pausa"	
Acción del Actor	**Respuesta del sistema**

5. El CneuroGameApp indica actualizar el administrador de estados.	5.1 El administrador de estados inicializa el estado pausa 5.2 El sistema muestra una ventana de confirmación preguntando si se desea retornar al menú principal.
Sección "Ejecutar estado salir"	
Acción del Actor	**Respuesta del sistema**
6. El CneuroGameApp indica actualizar el administrador de estados.	6.1 El administrador de estados inicializa el estado salir 6.2 El sistema muestra un video de salida y consecutivamente se abandona la aplicación finalizando así el caso de uso.
Sección "Ejecutar transiciones entre estados"	
Acción del Actor	**Respuesta del sistema**

7. El CneuroGameApp indica actualizar el administrador de estados.	7.1. El administrador de estados verifica si es necesario realizar un cambio de estado 7.2 En caso positivo se finaliza el estado actual y se inicializa el nuevo estado teniendo en cuenta el valor de control global de los estados ocurriendo de esta forma una transición entre estados .El sistema puede realizar las siguientes transiciones: a) Del estado inicio al estado menú. b) Del estado menú al estado jugar o al estado salir. c) Del estado jugar al estado pausa. d) Del estado pausa al estado jugar o al estado menú. 7.3 En el momento de la transición se actualiza el valor de control global de estados finalizando así el caso de uso.
Prioridad	Crítico

Tabla 2 Descripción del CU Manipular estados

Caso de Uso: 2	Crear perfil de jugador
Propósito	Permitir crear un perfil de jugador
Actores	jugador

Resumen:	El caso de uso se inicia cuando el jugador desea crear un perfil de jugador, inserta los datos necesarios (nombre, edad, nick, visión) para crearlo y finaliza con la creación del perfil.
Precondiciones	La ventana de perfil de jugador debe estar activa.
Poscondiciones	Se crea un perfil de jugador. Los datos insertados por el usuario se almacenan en un archivo txt.
Referencias	RF2

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
1. El usuario necesita crear un perfil de jugador y da clic en el botón cambiar jugador.	1.1 El sistema muestra la ventana de perfil de jugador con los datos a insertar (nombre, edad, nick, visión).
2. El jugador inserta los datos necesarios para crear un perfil de jugador (nombre, edad, Nick y visión, esta última puede ser OI u OD).	2.1 El sistema verifica que los campos estén lleno. 2.2 El sistema verifica que el nick no exista. 2.3 El sistema crea el perfil de jugador.
Curso alternativo	
	2.1 Si los campos están vacíos se emite el mensaje de advertencia “Campos vacíos”.
	2.2 Si el perfil ya existe se emite el mensaje “Perfil de jugador existente”, finalizando así el caso de uso.

Prioridad	Crítico

Tabla 3 Descripción del CU Crear perfil de jugador

Caso de Uso: 3	Configurar opciones de juego
Propósito	Permitir la configuración global del juego
Actores	Jugador
Resumen:	El caso de uso se inicia cuando el jugador decide configurar las opciones de juego, el actor personaliza las opciones de juego (pantalla completa, teclas a usar y volumen) acorde a su gusto y finaliza el caso de uso aplicando las modificaciones al juego.
Precondiciones	La ventana de configuración de las opciones de juego debe estar activa.
Poscondiciones	Se configuran las opciones de juego.
Referencias	RF3

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
1. El jugador desea configurar las opciones de juego y da clic en el botón opciones del menú principal	1.1 El sistema muestra la ventana de opciones de juego con los parámetros a modificar (pantalla completa, teclas a usar y volumen).
2. El jugador modifica las opciones de juego que desee (pantalla completa, volumen, teclas a usar).	2.1 El sistema aplica las modificaciones realizadas finalizando así el caso de uso.

Prioridad	Crítico

Tabla 4 Descripción de CU Configurar opciones de juego

Caso de Uso: 4	Iniciar juego
Propósito	Permitir dar inicio a la partida
Actores	Jugador
Resumen:	El caso de uso se inicia cuando el jugador decide comenzar el juego y culmina con el inicio de la partida.
Precondiciones	El menú principal del juego debe haberse cargado.
Poscondiciones	Se inicia la partida
Referencias	RF4
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
1. El jugador desea iniciar la acción del juego y presiona el botón “iniciar partida”.	1.1 El sistema carga los recursos necesarios para iniciar la partida y los inicializa(Lógica, Gráficos, Sonido ,IA) 1.2 El sistema inicia la partida finalizando así el caso de uso.
Prioridad	Crítico

Tabla 5 Descripción del CU Iniciar juego

Caso de Uso: 5	Salir del juego
Propósito	Permitir salir de la aplicación
Actores	Jugador
Resumen:	El caso de uso se inicia cuando el jugador decide salir del juego y culmina con el cierre de la aplicación.
Precondiciones	El menú principal del juego debe haberse cargado.
Poscondiciones	Se cierra la aplicación.
Referencias	RF4

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
1. El jugador desea abandonar la aplicación y presiona el botón “salir del juego”	1.1 El sistema cierra la aplicación finalizando así el caso de uso.
Prioridad	Crítico

Tabla 6 Descripción del CU salir del juego

Consideraciones Parciales

En este capítulo se lograron concretar las características del sistema; dígase requisitos funcionales y no funcionales, actores, casos de uso con sus pertinentes especificaciones y la relación entre dichos actores y casos de usos; lo más cercano posible a la conformidad del cliente lo que constituye un paso sustancial para el desarrollo exitoso de la siguiente etapa de esta investigación.

Están dadas las condiciones para comenzar el diseño e implementación del sistema. Se decidió prescindir del flujo de análisis debido a que no se está frente a un sistema complejo donde se necesite refinar requisitos, ni estructurar clases y paquetes del análisis como entrada al diseño para facilitar su ejecución .

Capítulo 3: Diseño e Implementación de la solución

3.1. Introducción

En el presente capítulo se diseña la propuesta de solución, haciendo uso de los patrones de diseño con el fin de lograr una solución refinada y elegante. Se obtiene el artefacto modelo de diseño con sus correspondientes diagramas de clases por paquetes del diseño y diagramas de secuencia por flujo de eventos, el cual constituye la entrada para la realización del modelo de implementación, donde se define la composición física del sistema reflejada en los diagramas de componentes fuertemente determinados por el lenguaje de programación empleado, culminando de este modo la construcción de la solución propuesta.

3.2. Modelo de diseño

El modelo del diseño es un modelo de objetos que describe la realización de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tiene impacto en el sistema a considerar, y constituye una entrada fundamental de las actividades de implementación.

El modelo de diseño del sistema que se desarrolla está compuesto por diagramas de interacción, específicamente de secuencia y por diagramas de clases organizadas en paquetes del diseño así como una breve descripción de dichas clases.

3.2.1. Patrones de diseño

Un patrón de diseño es básicamente una solución (un diseño) que surge de la experiencia práctica con varios proyectos, y que los equipos de desarrollo han encontrado que se pueden aplicar en diversos

contextos. Cada patrón de diseño describe a un conjunto de objetos y clases comunicadas. El conjunto se ajusta para resolver un problema de diseño en un contexto específico.

En el diseño de la aplicación se aplicó el patrón GOF de comportamiento estado, el cual se emplea cuando el comportamiento de un objeto cambia en dependencia del estado del mismo. Este patrón se puede apreciar mediante la clase abstracta CGameState que define las funciones básicas que realiza un estado de juego y en este caso sería el objeto que varía su comportamiento, lo que se manifiesta a través de las clases que son especializaciones de CGameState (CMenuState, CPlayState, PausaState, ExitState, InitState) y que implementan las responsabilidades específicas de cada estado de juego.

Además, se utiliza el patrón *Singlenton* (instancia única) el cual está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Se puede constatar concretamente en la clase *StateManager.* La aplicación solo necesita un administrador de estados que gestione las funciones pertinentes por lo que dicha clase implementa un método que devuelve la instancia única del *StateManager*.

3.2.2. Paquetes del diseño

Es una colección de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes que estén de alguna forma relacionados. Es usado para estructurar el modelo de diseño dividiéndolo en partes más pequeñas. Los paquetes de diseño deben usarse fundamentalmente como herramienta organizacional del modelo para agrupar elementos relacionados.(21)

Seguidamente se muestran los paquetes del diseño del sistema y sus relaciones.

Fig.13 Paquetes del diseño

Como lo ilustra la imagen el módulo se compone por tres paquetes, donde cada uno concentra clases con iguales funciones.

Paquete estados de juego: Agrupa las clases que implementan el caso de uso manipular estados, el cual encapsula toda la lógica del juego en estados de juego. Depende del paquete componentes de presentación para su funcionamiento.

Paquete componentes de presentación: Las clases que contiene, agrupan los casos de uso relacionados con la presentación del juego: Configurar opciones de juego, Jugar y salir. Depende del paquete *Scene Tool Kit* (STK) para su funcionamiento.

Paquete STK: Este paquete es un conjunto de clases que se emplean para facilitar la programación de los componentes de presentación del módulo: la clase Cbutton, Cpanel, Cedit y Ccheckbox.

- **Diagramas de clases del paquete estados de juego.**

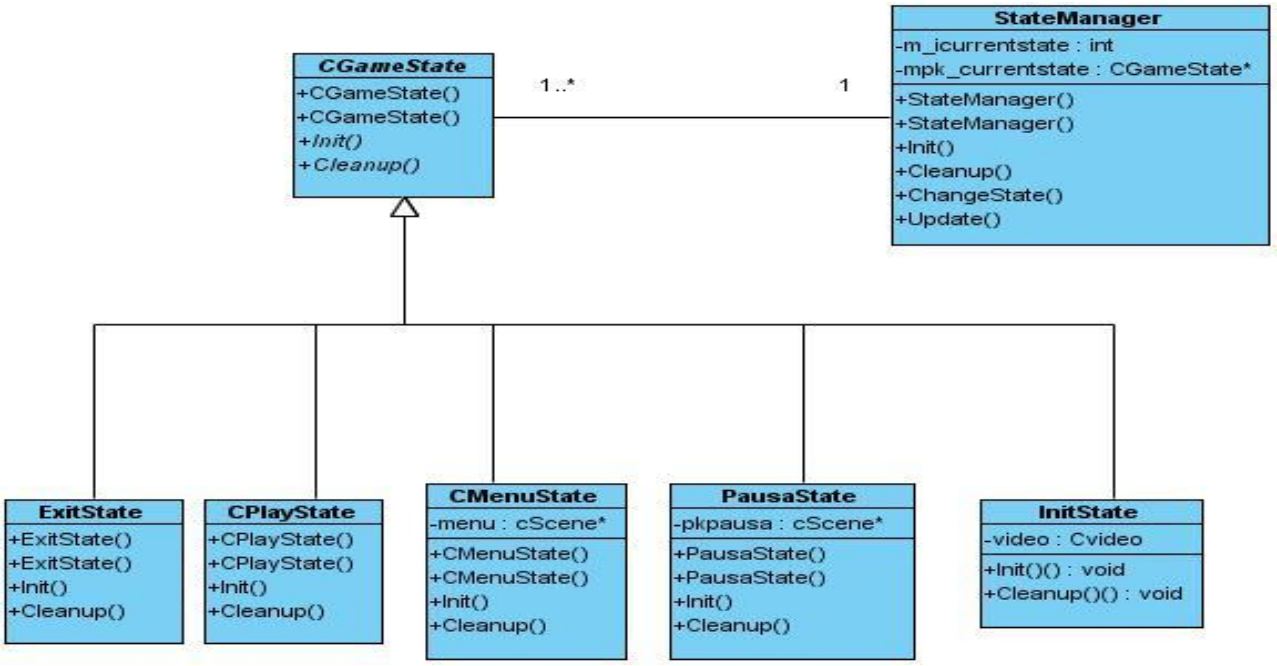
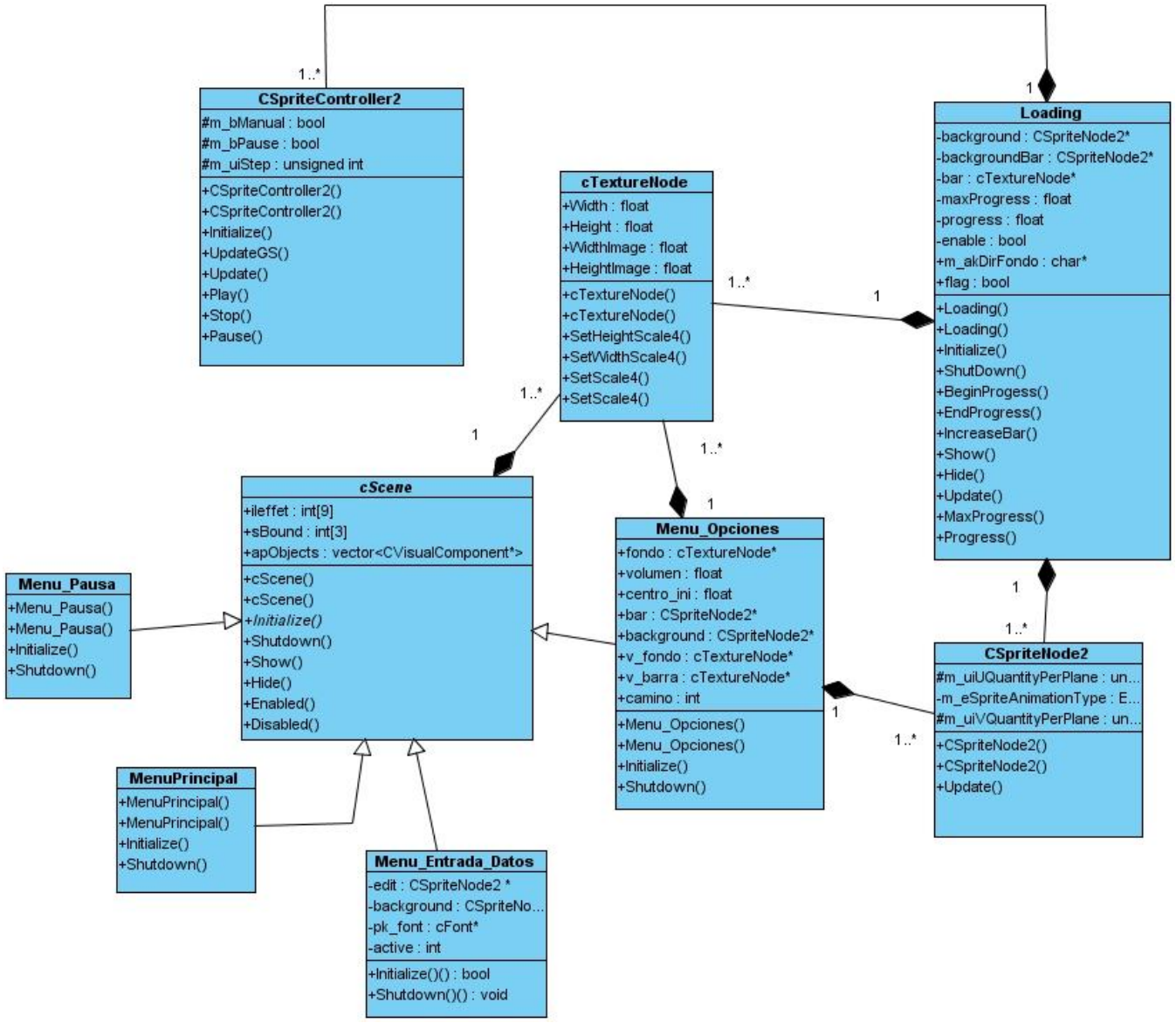


Fig.14 Diagrama de clases del diseño del paquete manipular estado

- **Diagramas de clases del paquete componentes de presentación**



- .

Fig. 15 Diagrama de clases del paquete componentes de presentación

- **Diagramas de clases del paquete STK.**

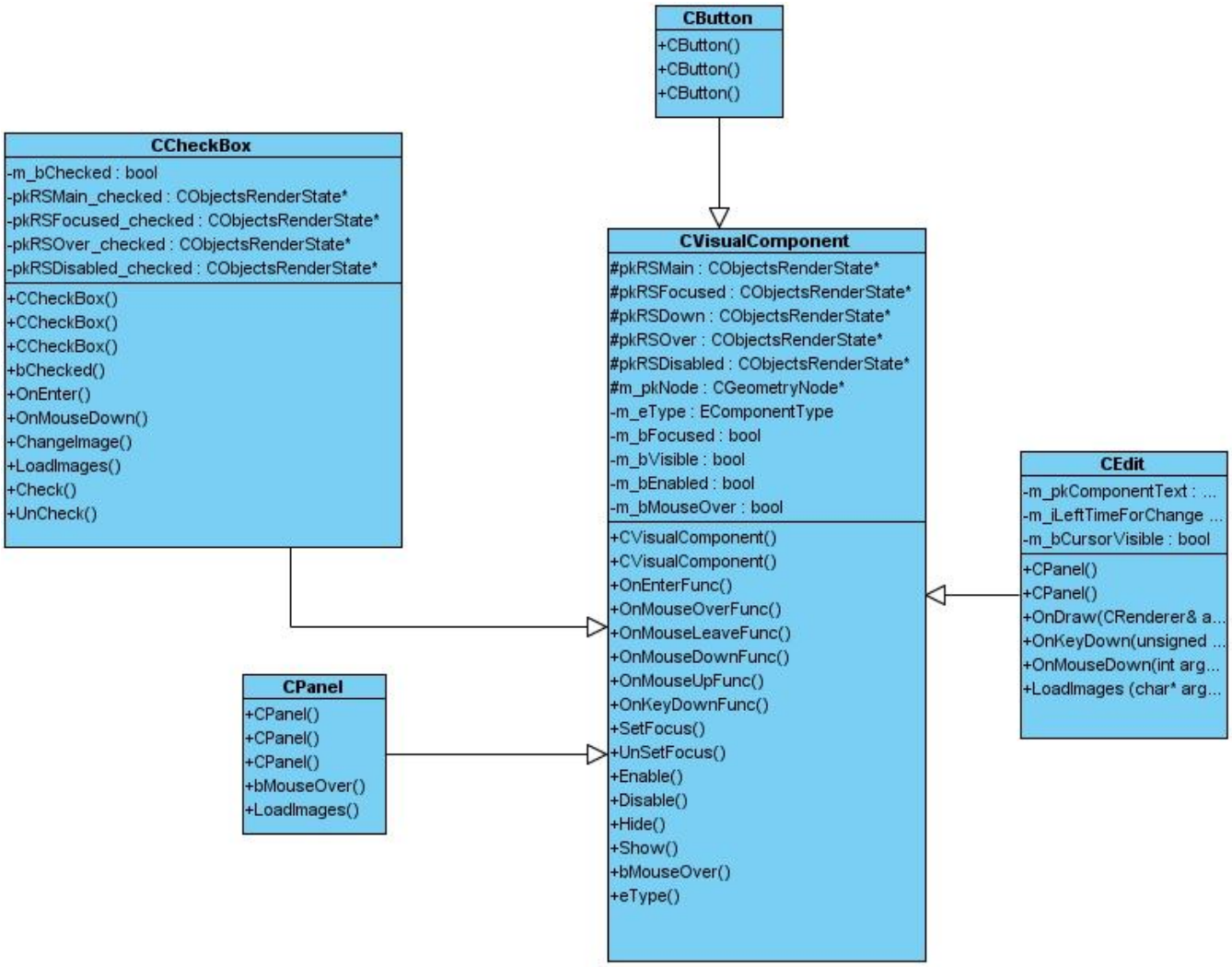


Fig. 16 Diagrama de clases del paquete STK

3.2.3. Diagramas de Secuencias

Los diagramas de secuencia y los diagramas de colaboración (ambos llamados diagramas de interacción) son dos de los cinco tipos de diagramas UML que se utilizan para modelar los aspectos dinámicos de los sistemas. Un diagrama de interacción muestra una interacción, que consiste en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos.

Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes. (20)

- **Diagrama de secuencia del CU "Manipular estados"**

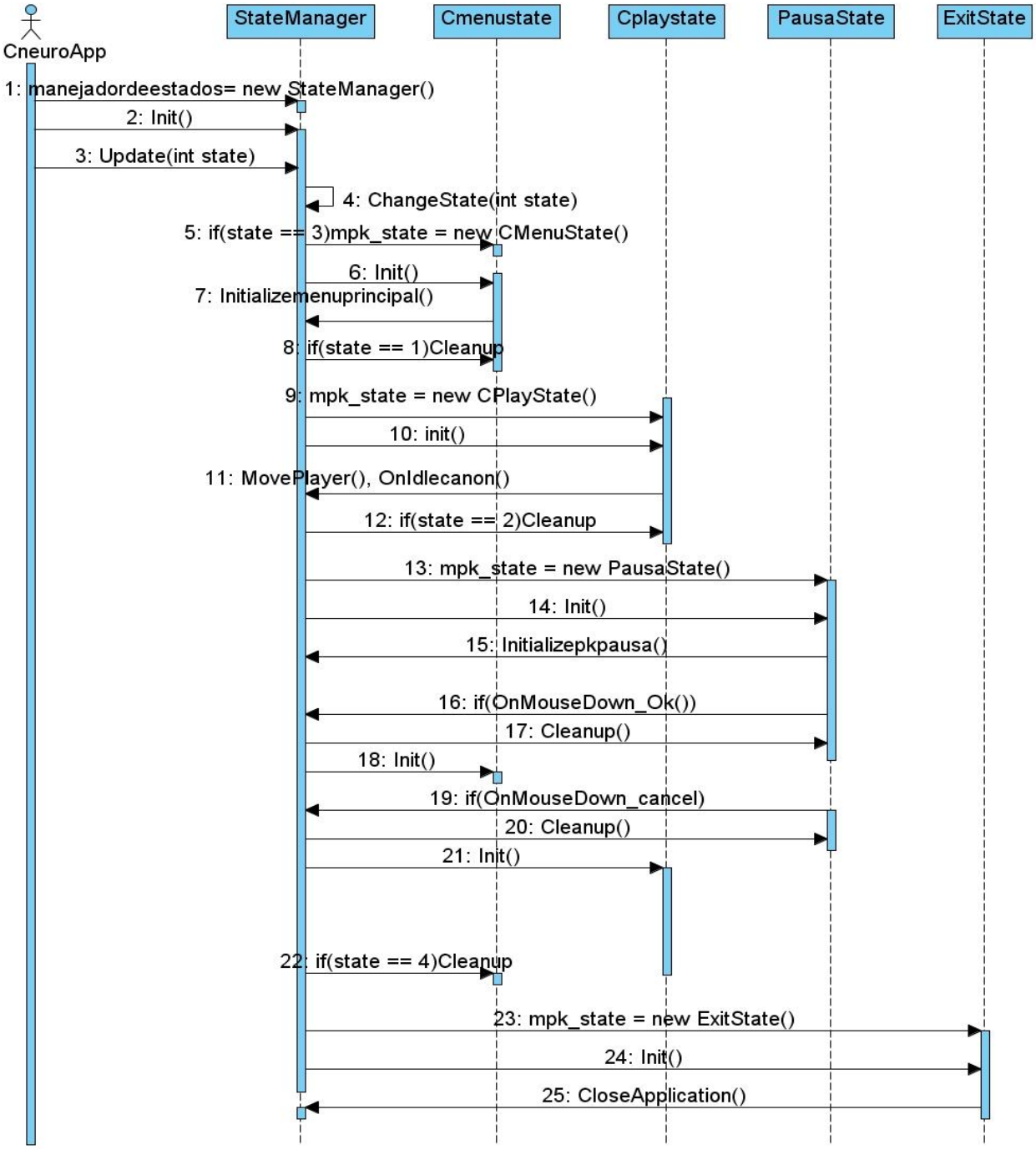


Fig. 17 Diagrama de secuencia del CU Manipular estados

- **Diagrama de secuencia del CU "Configurar opciones de juego"**

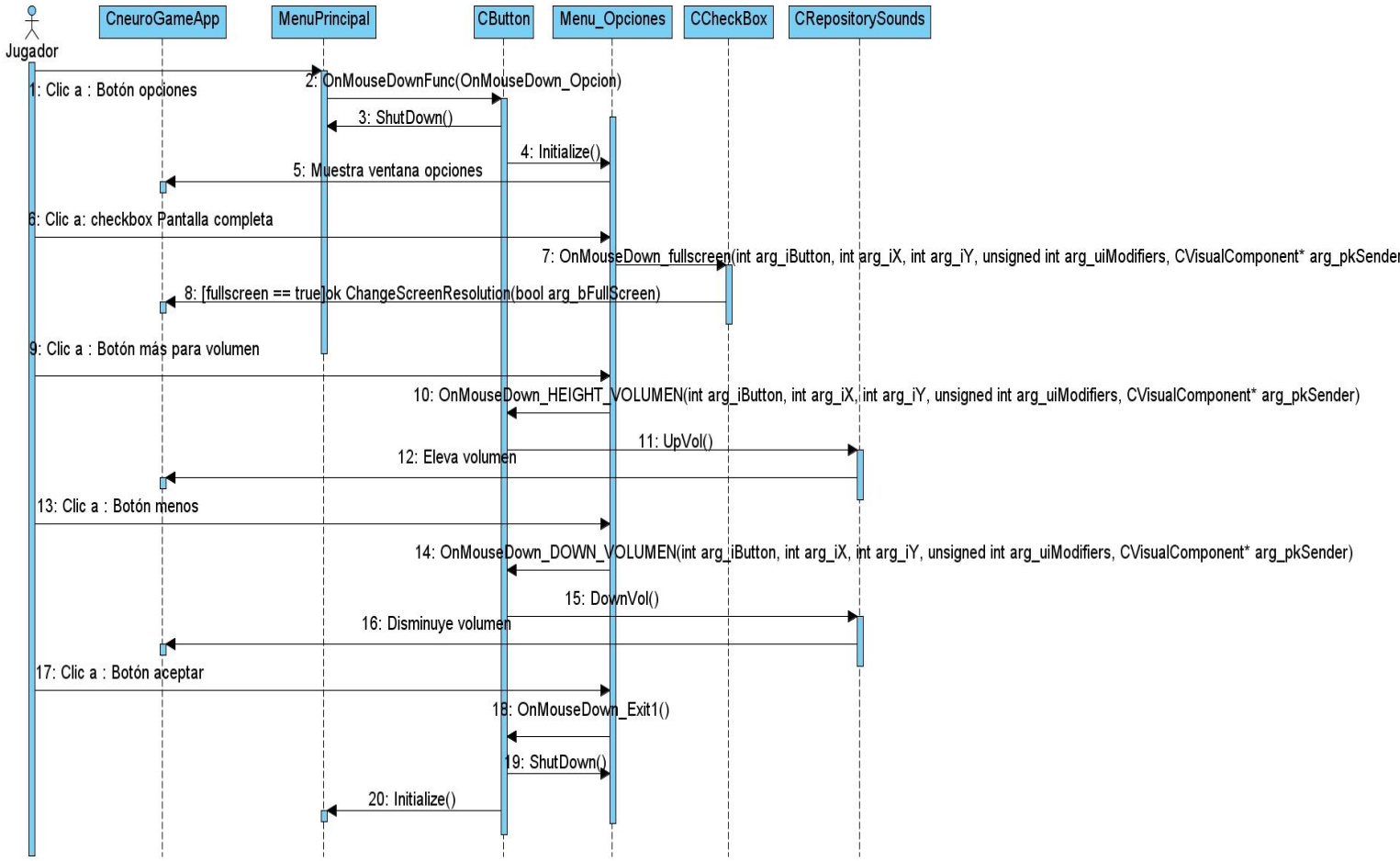


Fig. 18 Diagrama de secuencia "Configurar opciones de juego"

- **Diagrama de secuencia del CU "Iniciar juego"**

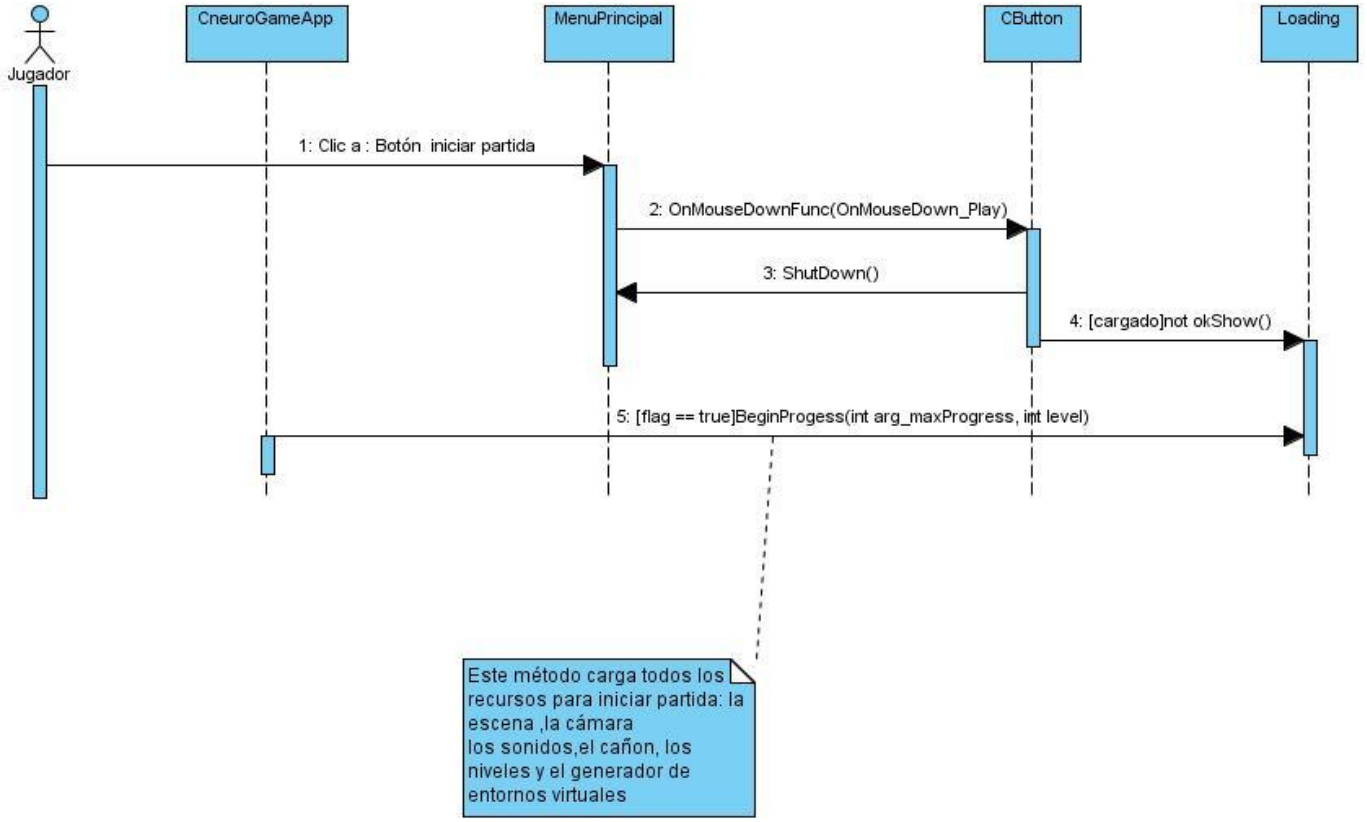


Fig. 19 Diagrama de secuencia del CU Iniciar juego

- **Diagrama de secuencia del CU "Salir del juego"**

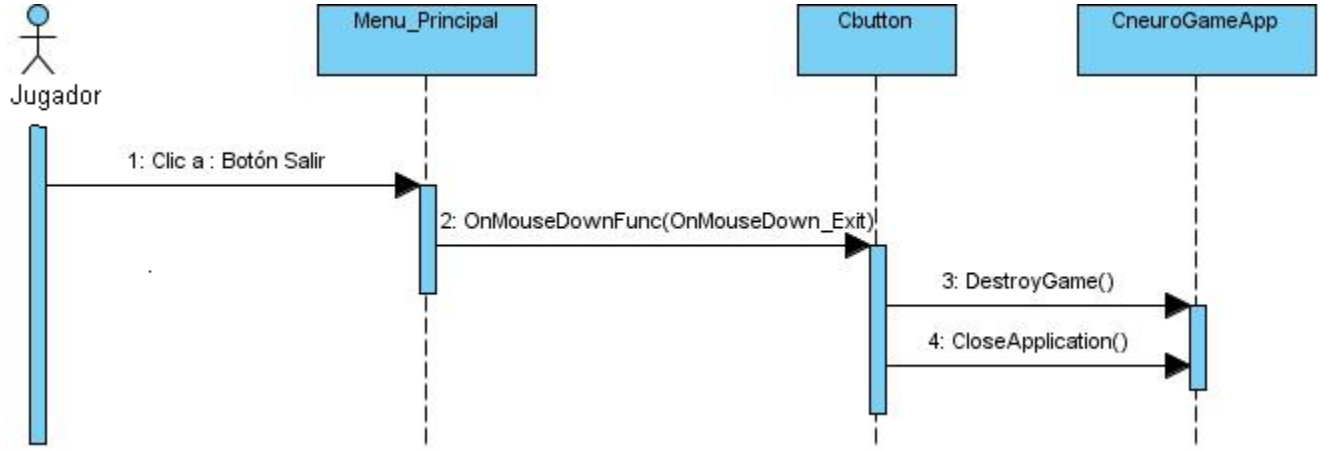


Fig. 20 Diagrama de secuencia del CU Iniciar juego

- **Diagrama de secuencia del CU "Crear perfil de jugador"**

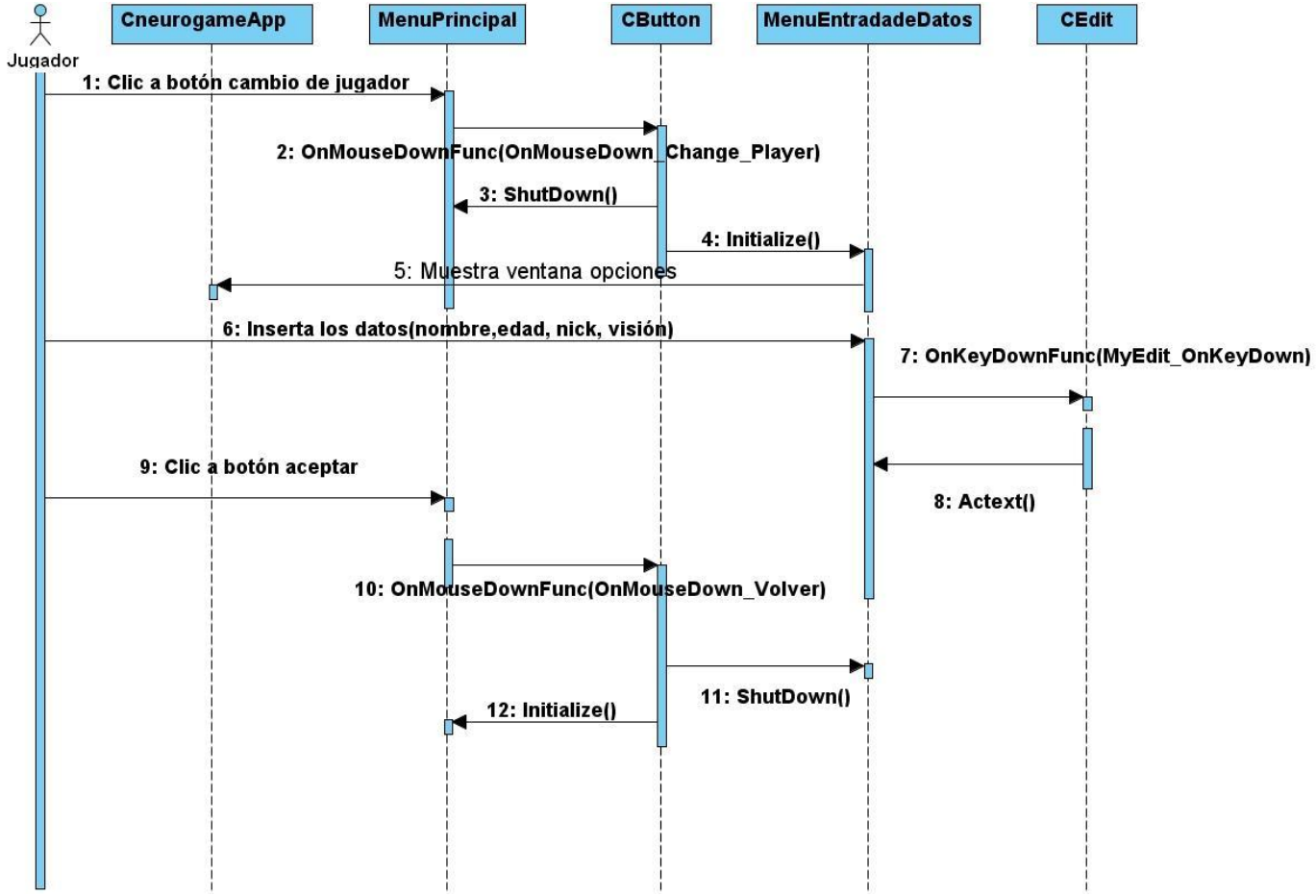


Fig. 21 Diagrama de secuencia del CU "Crear perfil de jugador"

3.2.4. Descripción de las clases de diseño

Nombre de la clase: CGameState		
Descripción: Clase genérica que define las funciones generales que posee un estado de juego.		
Atributos		
Tipo	**Nombre**	**Descripción**

virtual void	Init()	Método que inicializa el estado.
virtual void	Cleanup()	Método que destruye el estado.

Tabla 7 Descripción de la clase CGameState

Nombre de la clase: StateManager		
Descripción: Es la clase que administra los estados de juego y las transiciones entre ellos.		
Atributos		
Tipo	**Nombre**	**Descripción**
int	m_icurrentstate	Se usa para verificar que la variable de control de estados global haya modificado su valor
CGameState *	mpk_state	Representa el estado que debe inicializarse
CGameState *	mpk_currentstate	Representa el estado que debe destruirse

Métodos		
Tipo	**Nombre**	**Descripción**
void	Init()	Inicializa el administrador de estados
void	ChangeState()	Método más significativo del administrador de estados. Ejecuta la transición del estado actual hacia el estado que indique la variable de control de estados.
void	Update()	Actualiza el estado activo.

Tabla 8 Descripción de la clase StateManager

Nombre de la clase: Menu_Principal		
Descripción: Clase que implementa la ventana principal del juego**.** Define el menú principal del juego.		
Métodos		
Tipo	**Nombre**	**Descripción**
virtual bool	Initialize()	Método que inicializa el menú principal del juego. Se cargan los componentes visuales que lo forman: panel y botones.
virtual void	Shutdown()	Destruye cada componente del menú principal.

Tabla 9 Descripción de la clase Menu_Principal

Nombre de la clase: Menu_Opciones

Descripción: Clase que Implementa la ventana de opciones de juego.		
Atributos		
Tipo	**Nombre**	**Descripción**
int	camino	Se usa para actualizar la posición del apuntador del volumen en caso de haya sido modificada

Métodos		
Tipo	**Nombre**	**Descripción**
void	Initialize()	Método que inicializa la ventana de opciones de juego. Se cargan los componentes visuales que la forman: panel y botones.
void	ShutDown()	Destruye la ventana de opciones de juego.
void	OnMouseDown_HEIGHT_VOLUMEN (int arg_iButton, int arg_iX, int arg_iY, unsigned int arg_uiModifiers, CVisualComponent* arg_pkSender)	Eleva el volumen del juego en caso de que el botón más sea cliqueado .A su vez actualiza visualmente la posición del apuntador del volumen.
void	OnMouseDown_DOWN_VOLUMEN (int arg_iButton, int arg_iX, int arg_iY, unsigned int arg_uiModifiers, CVisualComponent* arg_pkSender)	Disminuye el volumen del juego en caso de que el botón más sea cliqueado .A su vez actualiza visualmente la posición del apuntador del volumen

Void	OnMouseDown_fullscreen(int arg_iButton, int arg_iX, int arg_iY, unsigned int arg_uiModifiers, CVisualComponent* arg_pkSender)	Modifica la resolución de la pantalla del juego si se marca el checkbox.

Tabla 10 Descripción de la clase Menu_Opciones

Nombre de la clase: Menu_Pausa		
Descripción: Implementa la ventana de confirmación que se muestra al pausar el juego.		
Métodos		
Tipo	**Nombre**	**Descripción**
void	Initialize()	Método que inicializa la ventana de confirmación. Se cargan los componentes visuales que la forman: panel y botones.
void	ShutDown()	Destruye la ventana de confirmación.

Tabla 11 Descripción de la clase Menu_Pausa

Nombre de la clase: Loading		
Descripción: Clase que implementa la carga de los recursos del videojuego para poder dar inicio a la partida. Durante el proceso de carga se muestra una barra que va escalándose proporcionalmente al avance de dicho proceso.		
Atributos		
Tipo	**Nombre**	**Descripción**

CSpriteNode2 *	background	Componente visual. Imagen de fondo de la barra que se muestra en el proceso de carga de la partida
CSpriteNode2 *	backgroundBar	Componente visual .Barra que se muestra en el proceso de carga de la partida.
cTextureNode *	bar	Componente visual .Color de la barra que se muestra en el proceso de carga de la partida.
float	maxProgress	Atributo que se usa para calcular la escala del progreso del color de la barra. Este progreso indica el porciento de carga de los recursos para el inicio de la partida.
float	progress	Posee el mismo objetivo del atributo anterior.
bool	enable	Indica si ha comenzado el progreso de la barra de carga.
bool	flag	Indica si la barra de carga esta activa

Métodos		
Tipo	**Nombre**	**Descripción**
void	Initialize()	Método que inicializa los componentes de la barra de carga: el fondo, el color y la barra en sí.

void	ShutDown()	Destruye los componentes de la barra de carga.
void	BeginProgess(int arg_maxProgress, int level)	Es el método fundamental en la clase. Se le pasa por parámetro el valor de la escala y el nivel a cargar. Inicializa la escena, el nivel de juego, el cañón y el generador de entornos virtuales y paralelamente va escalando la barra de carga.
void	EndProgress()	Finaliza la escala de la barra de carga
void	IncreaseBar(float arg_value)	Calcula la dimensión a la que debe ser escalada la barra y la pinta en la nueva posición.
void	Show()	Muestra la barra de carga
void	Hide()	Oculta la barra de carga

Tabla 12 Descripción de la clase Loading

Nombre de la clase: CMenuState		
Descripción: Implementa las funciones del estado menú		
Atributos		
Tipo	**Nombre**	**Descripción**

cScene *	menú	Define un objeto polimórfico de tipo menú principal
Métodos		
Tipo	**Nombre**	**Descripción**
void	Init()	Inicializa el estado menú cargando el menú principal del juego.
void	Cleanup()	Destruye el estado menú, eliminando el menú principal del juego.

Tabla 13 Descripción de la clase CMenuState

Nombre de la clase: PausaState		
Descripción: Implementa las funciones del estado pausa.		
Atributos		
Tipo	**Nombre**	**Descripción**
cScene *	pkpausa	Define un objeto polimórfico de tipo menú pausa
Métodos		
Tipo	**Nombre**	**Descripción**
void	Init()	Inicializa el estado pausa cargando la ventana de confirmación que se muestra cuando el juego está pausado

void	Cleanup()	Destruye el estado pausa eliminando la ventana de confirmación

Tabla 14 Descripción de la clase PausaState

Nombre de la clase: CPlayState		
Descripción: Implementa las funciones del estado play		
Métodos		
Tipo	**Nombre**	**Descripción**
void	Init()	Activa las funcionalidades del cañón (movimiento y tiro) y las del generador de entornos.
void	Cleanup()	Destruye la partida culminando de esta forma el estado jugar

Tabla 15 Descripción de la clase CPlayState

Nombre de la clase: ExitState		
Descripción: Implementa las funciones del estado pausa.		
Métodos		
Tipo	**Nombre**	**Descripción**

void	Init()	Inicializa el estado salir mostrando un video de salida y finalizando la aplicación

Tabla 16 Descripción de la clase ExitState

3.3. Modelo de Implementación

El modelo de implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se encuentran datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos. Este artefacto describe cómo se implementan los componentes, congregándolos en subsistemas organizados en capas y jerarquías, y señala las dependencias entre éstos. (22)

Teniendo en cuenta la entrada del modelo de diseño y la traza directa de este con el modelo de implementación, los diagramas de componentes se estructuraron por paquetes. El paquete Manipular estados agrupa los componentes que implementan la estructuración de la lógica del videojuego en estados de juego. El paquete componentes de presentación describe como se implementan en términos de componentes los elementos de la interfaz visual del videojuego.

3.3.1. Diagrama de componentes del paquete manipular estados

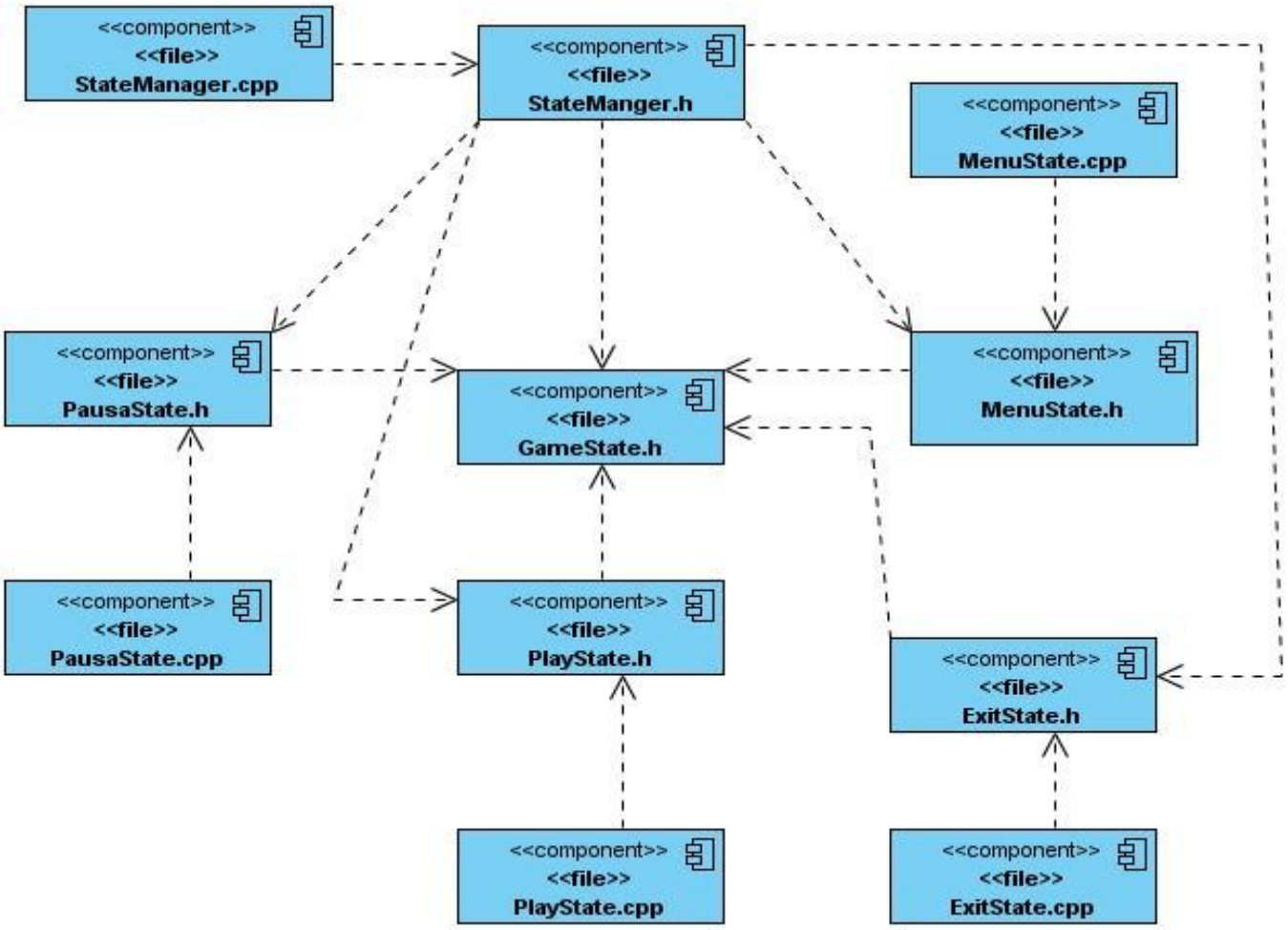


Fig. 22 Diagrama de componentes del paquete de estados

3.3.2. Diagrama de componentes del paquete componentes de presentación

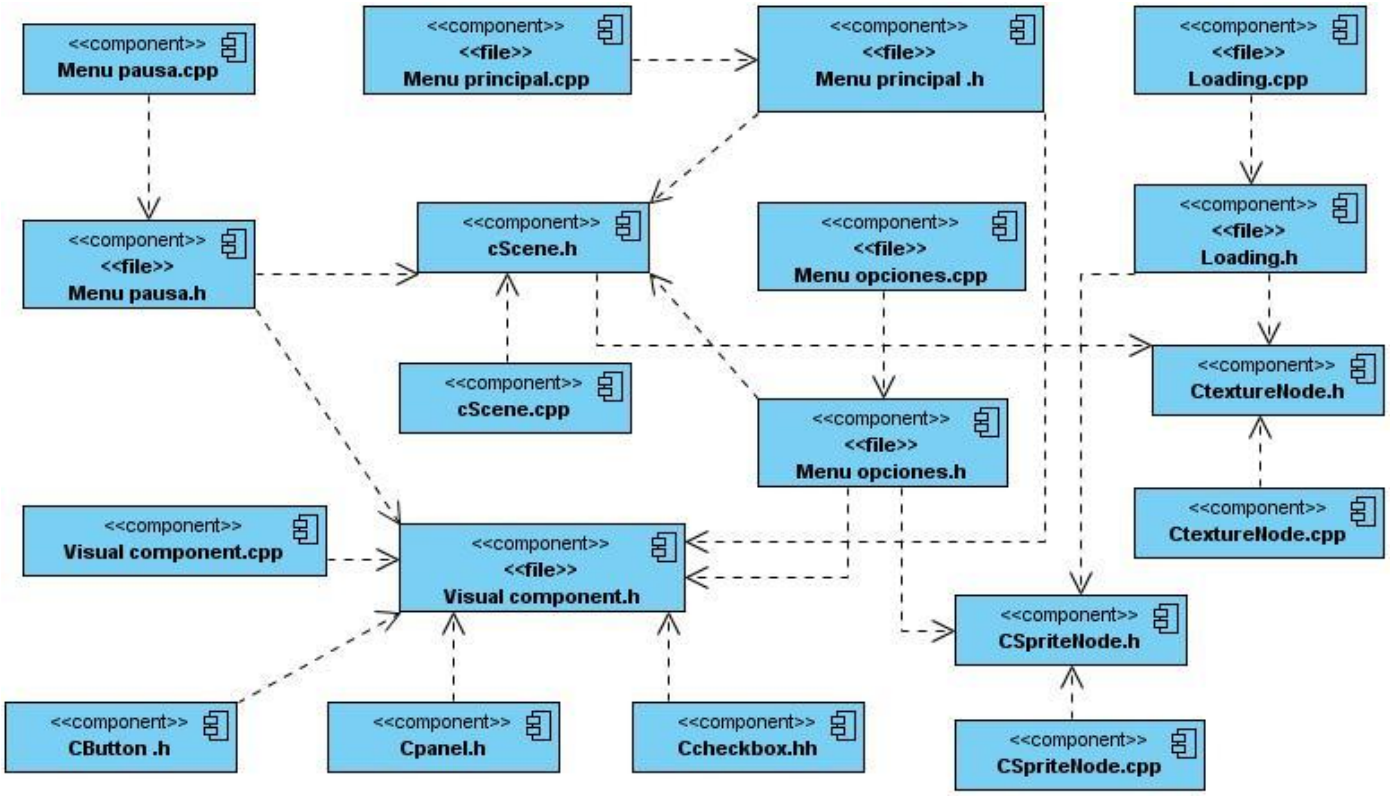


Fig. 23 Diagrama de componentes del paquete componentes de presentación

Consideraciones Parciales

En este capítulo se modeló el diseño con sus diagramas de clases correspondientes y diagramas de secuencia por flujo de eventos. El resultado del diseño sirvió de entrada para la realización del flujo de trabajo de implementación donde a partir de los diagramas de componentes realizados agrupando los

casos de uso por paquetes de funcionalidades, se implementó el sistema en términos de componentes siendo determinado fuertemente por el lenguaje de programación usado para la construcción del mismo, lográndose además de la aplicación, la documentación necesaria que describe el proceso ingenieril llevado a cabo a lo largo del ciclo completo de desarrollo del software.

Conclusiones Generales

Al término del presente trabajo, se obtiene como resultado una aplicación que resume la lógica del funcionamiento de los estados de juego y la creación de una presentación coherente y de fácil comprensión para el jugador. Al resultado alcanzado lo precedió una amplia investigación en pos de comprender y extractar la esencia teórica de los estados de juego e interfaz gráfica de usuario, la forma de aplicarlos en motores de videojuegos y las buenas prácticas de las que se pueden hacer uso en este sentido; todo ello con el fin de determinar la vía más adecuada a ser empleada en la construcción de la solución.

Se demostró que el gestor de estados logró dividir con eficacia las funcionalidades del motor de videojuegos Cneuro Game Engine, concentrando específicamente en cada estado de juego las responsabilidades inherentes a él, poniéndose de manifiesto el principio de la programación de la modularidad pues cada estado resuelve un fragmento del problema gestionando luego el administrador de estados la interacción entre ellos. Este sistema permitió reducir la complejidad de la implementación del CneuroGameApp, pues muchas de las tareas que allí se trataban, pasaron a ser manipuladas por los diferentes estados.

Se integró con éxito el modulo de estados de juego y componentes de presentación al motor de videojuego Cneuro Game Engine, brindándole funcionalidades esenciales para el alcance de un producto con calidad.

Se logró brindar una factible navegabilidad al videojuego Meteorix a través de sus diferentes ventanas de juego permitiendo de esta manera que sea manipulado con mayor facilidad. Se diseñó una interfaz gráfica sencilla, comprensible y usable que torna la relación usuario-juego mucho más amigable y por lo tanto posibilita una mayor rapidez en la realización de las operaciones con el *software*.

Las funcionalidades del módulo de estados de juego y componentes de presentación se traducen en satisfacción, positiva apreciación y facilidad de uso para los usuarios finales de los videojuegos construidos con el Cneuro Game Engine, que en estos momentos son los niños con afecciones de ambliopía que usarán el Meteorix en su rehabilitación visual.

Recomendaciones

Con la finalidad de potenciar una aplicación gradualmente superior y más completa en términos de eficiencia y funcionamiento se recomiendan los siguientes aspectos:

- Adicionar al módulo nuevos estados de juego que posibiliten emplear un rango de estados más amplio y flexible en la implementación del gestor y de esta forma poder suplir la mayor parte de las necesidades que se demanden.
- Incorporación de nuevos componentes visuales que enriquezcan la interfaz gráfica de usuario y la tornen más intuitiva y usable al jugador, como por ejemplo la lista de elementos.

Bibliografía Referenciada

1. **García García, Inmaculada**. *SIMULACIÓN HÍBRIDA COMO NÚCLEO DE SIMULACIÓN .* Universidad de Valencia , Septiembre del 2004: Tesis Doctoral Inédita.
2. Alonso Alonso, J. *Videojuegos 3D.* Universidad Abierta de Cataluña.
3. **Duch i Gavaldà, Jordi . Tejedor Navarro ,Heliodoro.** *Diseño y programación de videojuegos.Lógica de un videojuego.* Universidad Abierta de Cataluña
4. **Marrero Expósito, Carlos.** (2006). *Mente Expansiva. Interfaz Gráfica de Usuario: aproximación semio-cognitiva.* [Citado el el 4 de Diciembre de 2009] http://www.chr5.com/investigacion/investiga_igu/index_igu.html.
5. **Aimacaña Toledo, Carlos.** (s.f.). *wikiciencia. Interfaz de usuario.* [Citado el 20 de noviembre de 2009] http://www.wikiciencia.org/informatica/programacion/iusuario/index.php.
6. **Mercovich, Eduardo**. (10 de 2 de 2004). *Ponencia sobre Diseño de Interfaces y Usabilidad: cómo hacer productos más útiles, eficientes yseductores.* [Citado el 20 del 2 del 2010] http://planeta.gaiasur.com.ar/infoteca/siggraph99/diseno-de-interfaces-y-usabilidad.html
7. **Duch i Gavaldà, J. N.** *Sonido ,Interación y redes.* Universidad Abierta de Cataluña.
8. **Peinado Santorum, F. M.** (2006). De cómo la realidad puede tomar parte en juegos emergentes– REVISTA DE COMUNICACIÓN Y NUEVAS TECNOLOGÍAS. (Nº 8).
9. **AmericaTI EIRL**.Ventajas y desentajas.Comparación de los lenguajes c, c++ y Java. (11 de 11 de 2006). *Americati.* [Citado el 21 de 1 de 2010] http://www.americati.com/doc/ventajas_c/ventajas_c.html
10. **Prieto., A. C.** (2009.). *Ingeniería Inversa a Biblioteca de Inteligencia Artificial para videojuegos.* La Habana: Trabajo de Diploma en opción al título de ingeniero en ciencias infórmaticas, UCI.
11. **UCI.** (2009.). *Fase de Inicio. Disciplina de Requisitos. Ingeniería de software 1.* [Citado e15 de 3 de 2010] EVA: http://eva.uci.cu/mod/resource/view.php?id=22095

12. **UCI.** Disciplina de Análisis y Diseño. (26 de 2 de 2010). *EVA.* [Citado el 2 de 3 de 2010] http://eva.uci.cu/mod/resource/view.php?id=14069%29.

13. **Maribel Silva Muñoz, Sándor Rodríguez Prieto**. *Diseño e Implementación de un sistema informático integrado para la Gestión de Compras de Bienes y Contratación de Servicios en los Registros y las Notarías de la República Bolivariana de Venezuela.* La Habana: s.n., 2008

Bibliografía Consultada

1. Game Dev Geek. Managing Game States in C++[Citado el 20/3/2010]. http://gamedevgeek.com/tutorials/managing-game-states-in-c/
2. **Fox, Brent.** *Game Interface Design.* Boston : s.n. 1-59200-593-4.
3. **Alvarez Morales, Pablo.Escudero Sánchez, Eduardo.** *Programación orientada a objetos en los videojuegos.* s.l. : Departamento de informática y Automática .Universidad de Salamanca.
4. **López Victor, Francisco Montero,María Lozano.** *Virtual- Prismaker: Ayudando en elproceso de aprendizaje.Juego Físico vs juego virtual.Un solo juego dos modos de interacción.* Universidad de Castilla - La Mancha.
5. **Fernández Leiv, Dr. Antonio J.a . Guerrero García Dr. Pablo.** *I jornada de alumnos de infórmaticas sobre juegos :Matemática recreativa eimplementación de videojuegos.* Málaga . 2008.
6. **Acevedo Moreno, David Aurelio.** *Diseño de una arquitectura para incorporar emociones en vidoejuegos.* México , D.F : tesis de maestría, 2009.
7. **Acerenza, Nicolás. Coppes Ariel .Mesa ,Gustavo .Viera ,Alejandro .Fernández,Eduardo.** *Una metodología para desarrollo de videojuegos: versión extendida.* Montevideo, Uruguay : Universidad de la República, 2009. 0797-6410.
8. **J. L. González Sánchez, N. Padilla Zea, F. L. Gutiérrez, M. J. Cabrera.** *De la Usabilidad a la Jugabilidad: Diseño de Videojuegos Centrado en el Jugador.* España : Universidad de Granada.
9. **Duch i Gavaldà, Tejedor Navarro , Heliodoro.** *Introducción a los videojuegos.* s.l. : Universidad Abierta de Cataluña.
10. **Jesús, Alonso Alonso.** *Videojuegos 2D.* s.l. : Universidad abierta de Cataluña.
11. **---.JACOBSON, Ivar, RUMBAUGH, James and BOOCH, Grady.** *El proceso unificado de desarrollo.* s.l. : Addison Wesley., 2000.

12. **Rose2003), Rational Unified Process** *(ayuda de la Suite de Rational).*

13. **MARCANO LÁREZ, Beatriz (2006):** *Estimulación emocional de los videojuegos: efectos en el aprendizaje. En GARCÍA CARRASCO, Joaquín (Coord.) Estudio de los comportamientos emocionales en la red [monográfico en línea].* Revista electrónica Teoría de la Educación: Educación y Cultura en la sociedad de la información. Vol. 7, nº 2. Universidad de Salamanca. [Fecha de consulta: 15 /01/2010]. http://www.usal.es/~teoriaeducacion/rev_numero_07_02/n7_02_beatriz_marcano.pdf *ISSN 1138-9737*.

Glosario de términos general

Ambliopía: Disminución de la agudeza visual en uno o ambos ojos.

Apreciación: Es una medida de las percepciones, opiniones, sentimientos y actitudes generadas en el usuario por la herramienta o sistema; una medida, si se quiere, de su seducción o elegancia.

Jugabilidad: Conjunto de propiedades que describen la experiencia del jugador ante un sistema de juego determinado: Satisfacción, aprendizaje, eficiencia y efectividad, motivación, inmersión, emoción y socialización.

Meteorix: Nombre del videojuego para el tratamiento terapéutico de la ambliopía desarrollado por el proyecto Juegos Cneuro.

Navegabilidad: Facilidad con la que el jugador puede transitar a través de las ventanas o pantallas del videojuego.

Overhead: Tiempo desperdiciado por el procesador para realizar un cambio de contexto.

Presentación: Conjunto de componentes visuales, que se muestran antes del comienzo de la partida y que le dan introducción al juego.

Renderizar: Crear en forma automática una imagen de acuerdo al modelo tridimensional que existe en el ordenador.

Usabilidad de un videojuego: Medida de su utilidad, facilidad de uso, facilidad de aprendizaje y apreciación para una tarea, un usuario un contexto dado.

Glosario de abreviaturas:

BSP: *Binary Space Partitioning* (Particionamiento binario del espacio).

CASE: *Computer Aided Software Engineering* (Ingeniería de software asistida por ordenador).

CNEURO: Centro de Neurociencia.

IA: Inteligencia Artificial.

IDE: *Integrated Development Enviroment (Entorno de desarrollo Integrado).*

GOF: *Gang Of Four (*Grupo de los cuatros*)*

ODE: *Open Dynamics Engine* (Biblioteca para la simulación dinámica y la detección de colisiones).

RUP: *Rational Unified Process.* (Proceso Unificado del Rational).

STK: Scene Toolkit (Motor gráfico).

UML: *Unified Modeling Language* (Lenguaje Unificado de Modelado).

Anexo 1

Fig. 24 Vista del estado menú

Anexo 2

Fig 25 Vista del estado jugar

Anexo 3

Fig 26 Vista del estado pausa